

THE ARCHITECTURE SERIES, DECADE BY DECADE

From Terminal to Agent

Software Architecture, Decade by Decade

Every decade's architecture was a sound answer to the thing that ran out first: machine time, the complexity of large projects, the ability to change a program, the application backlog, integration, team speed, coordination between teams, and now trust in work that AI made. This book follows one insurer's claims system as it grows from punched cards in 1956 to AI agents in 2026, and gives a CTO the Decision Card: a plain way to record what a design decides, where its boundaries are, and how it would fail.

Juan Cardena

Enterprise AI & Agentic Systems Architect · blog.jcardena.com

CONTENTS

BEFORE YOU START

Reader's Brief: The Binding Constraint and Six Questions

PROLOGUE

One Claim, Seventy Years

PART I · THE LENS

01 What Architecture Decides

02 The Binding Constraint, and How to Read Old Designs

PART II · THE DECADES

03 Before 1960: Wiring the Machine

04 The 1960s: The Terminal and the Crisis

05 The 1970s: Structure and Change

06 The 1980s: Computing Leaves the Glass Room

07 The 1990s: Integration and the Web

08 The 2000s: Services and the Team

09 The 2010s: Many Teams, One Platform

10 The 2020s (Our View): When AI Writes the Code

PART III · WHAT CHANGED

11 What Returned, and What Did Not Carry Over

12 One Claims System Today: Keep, Wrap or Replace

CLOSING

Epilogue: Designed to Run, and the Next Binding Constraint

APPENDICES

A The Decision Card: Blank and Worked

B Timeline, 1945 to 2026

C Glossary

D Platform Mapping, as of November 2026

E Sources

BEFORE YOU START

Reader's Brief: The Binding Constraint and Six Questions



This book reads seventy years of software architecture through one idea: in every decade, something ran out first, and the designs of that decade were answers to it. It follows one invented insurer's claims system from punched cards in 1956 to AI agents in 2026, and it gives you one tool, the Decision Card, to record what a design decides. This brief is the whole method in a few pages. The rest of the book is the evidence.

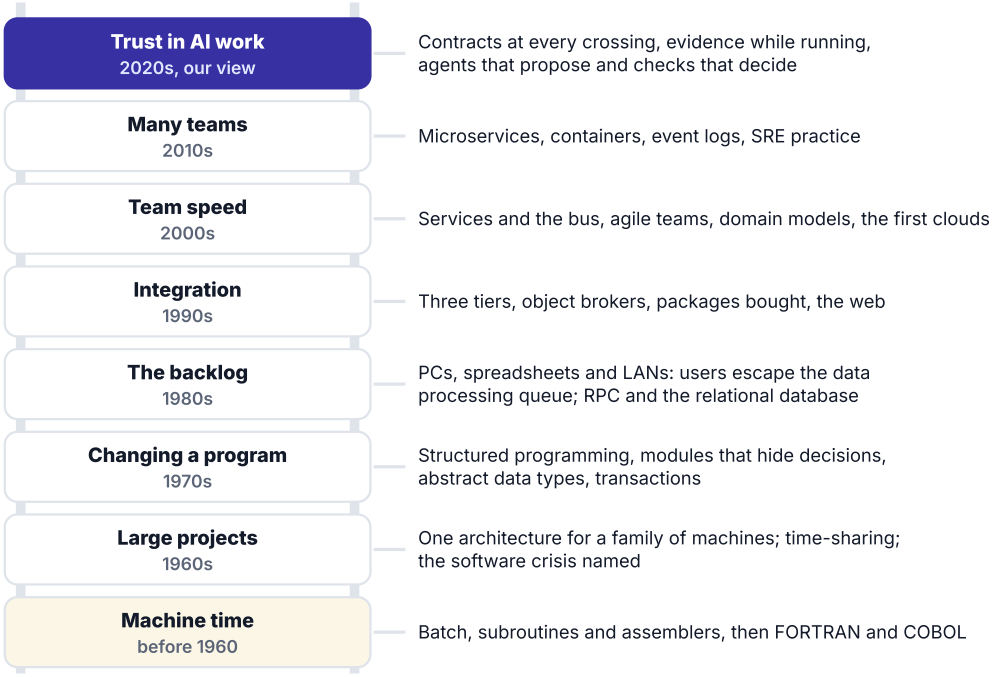
The binding constraint

A **binding constraint** is the resource that runs out first and so limits everything else. Before 1960 it was machine time; in the 1960s, the complexity of large projects; in the 1970s, the cost of changing a program; in the 1980s, the backlog of applications users waited for; in the 1990s, integration; in the 2000s, the speed of a team; in the 2010s, coordination between many teams; and in the 2020s, in this book's view, trust in work that AI made. Figure R.1 shows the ladder; Chapter 2 tests it, gives each rung a second constraint, and says where the frame fails.

The idea is not new. Eliyahu Goldratt's business novel *The Goal*, written with Jeff Cox and first published by North River Press in 1984, introduced the theory of constraints: a system's output is set by its constraint, so improve that first, then look for the next one. Goldratt later set the method out as five focusing steps, which the Theory of Constraints International Certification Organization dates to 1987; the first is "Identify the system's constraints." This book borrows the idea and applies it to architecture by decade. The test it gives you is simple: judge a design by the constraint of its own era before you judge it by today's.

FIGURE R.1

The binding constraint, decade by decade: what ran out first at the leading edge



Each rung names what ran out first for the firms at the leading edge of that decade; most companies, the insurer included, met each rung years later.

Author's synthesis; Chapter 2 gives the secondary constraints and the indicators.

Eight principles, and where they came from

The author's principles are the lens of this series. They appear in full only here. Each book leads with two or three of them; this one leads with P2, P5 and P3. In the chapters, the author's own view is kept in a box marked "Juan's lens", at most one per chapter, and a judgment or forecast that is not settled history goes in a box marked "Our view, as of November 2026", so that the narrative history stays neutral.

#	Principle	Ancestors
P1	Start from the business outcome, then the guarantees	Deming's 14 Points for Management, set out in *Out of the Crisis*
P2	Every crossing is a boundary; put a contract on every door	Parnas, information hiding (CACM, December 1972); Meyer, "Applying 'Design by Contract'" (IEEE Computer, October 1992)
P3	Evidence, not assertions	The control totals of the punched-card era (Chapter 3), which showed that a batch was complete and that its totals agreed before anything was paid
P4	Four guarantee layers: quality, security, visibility, cost	The quality models of ISO/IEC 25010 (2011, revised 2023), which count security as one quality characteristic; separate visibility and cost layers are the author's addition
P5	Software must not fail silently	Gray, "Why Do Computers Stop and What Can Be Done About It?" (Tandem, June 1985): a module either works or stops
P6	Security is an operating loop, not a gate	Saltzer and Schroeder's design principles (1975); the NIST Cybersecurity Framework (February 2014), whose functions are "concurrent and continuous"
P7	Design for the run: a small team plus agents operates it	Site reliability engineering, as described in Google's *Site Reliability Engineering*, edited by Beyer, Jones, Petoff and Murphy
P8	The model proposes; the contract decides	The author's own method, set out in *The Contract Decides*

Six questions and one card

Every chapter asks the same six questions of the designs it describes:

- What outcome does it serve?
- Where are its boundaries?
- Who can change it?
- How does it fail, and what would show us?
- What does it cost to run and to change?
- What assumption, if it changed, would make it wrong?

The **Decision Card** puts the answers on one page. Its header gives an id, a date, a status, an owner (a job title, never a person's name) and the card it supersedes. Ten fields follow: the decision in one sentence; the outcome and the binding constraint; the boundaries and what crosses each one; the options rejected; one line each on quality, security, visibility and cost; how it fails and how we would see it; how it runs; what reversing it would cost; its assumptions and the trigger to revisit it; and a change guide, the things any changer, person or agent, must not break. A status is one of Proposed, Accepted, Rejected, Superseded, Reversed or Retired. Each decade chapter carries a worked card for the insurer, and Appendix A has the blank card, a guide to each field, and the life of one decision from 1974 to today.

Two kinds of year

This book uses two kinds of date, and the difference matters. A **release year** is when a vendor or a standards body made something available: IBM first shipped CICS, as PU-CICS, in 1968, and the CICS program product followed on 8 July 1969. An **adoption year** is when the insurer started to use it: it put its claims inquiry on CICS in 1982. Every year in the insurer's story is an adoption year. Where both kinds of year could be confused, the text gives both, and Chapter 2 and Figure 2.3 set the pairs side by side.

The insurer is invented and illustrative. It was founded in 1898, and every practice it follows is one that real companies of the period followed, with the source given. The people and the single incidents in this book are invented too, and stay inside it.

SECURITY LENS

Who can change it. In every decade, the first question about a design's security is who is able to change the thing, and whether anyone would know. **How fast it is fixed.** The second is how long a known weakness stays open, which depends on how the system is built and run, not only on the fix. **What AI changes.** When agents can write and run code, the number of possible changers grows fast, so the card asks for both answers: who can change it in its security line, and how fast a fix ships in "how it runs".

Monday rule: pick one system you inherited, name the constraint it was built against, and ask whether that constraint still binds before you call the design wrong.

PROLOGUE

One Claim, Seventy Years



Two fires break out seventy years apart, and the same insurer pays for both. The first claim, in 1956, travels on paper and punched cards and ends with a paper check. The second, in 2026, is reported on a website at night, read by software agents within minutes, and written into the insurer's books by a batch program first adopted in 1974. Following both claims is the quickest way to see what this book is about: a claims system whose core has not been rebuilt since 1974, only added to, and the boundaries that kept it working.

1956: a claim on cards

The insurer is invented. It was founded in 1898, writes home, commercial, motor and liability cover, and does what mid-size American insurers of each period did. Its policy records have been on punched cards since 1931. In 1956 it begins to tabulate claims on cards too.

A family's kitchen burns on a winter evening. The next morning the policyholder telephones the branch office, and a clerk writes the details on a claim form: the policy number, the date of the loss, the cause, an address. An adjuster, the person who inspects a loss and agrees its value, visits the house that week, looks at the damage, and writes a loss report with an agreed amount.

The form and the report go to the card room. A key-punch operator types the claim onto an 80-column card. Which columns hold the claim number, which hold the amount, and which hold the date is fixed in a card layout that every later step relies on. At the end of the day the cards are sorted and run through a tabulator, an electromechanical accounting machine, such as IBM's 407, whose processing is set by a wired control panel, a plug board; the shop keeps a wired panel for each job and inserts the one it needs. It prints a claims register and a total.

The insurer's safeguard is a control total, a practice of the period that Chapter 3 sources. Before the cards go in, a clerk adds up the amounts on the forms on an adding machine. If the machine's total differs from the clerk's, the batch stops until someone finds the card that

is wrong. Nothing is paid until the totals agree. Then a check is typed, signed and posted to the family. The whole journey takes days and is visible at every step: a person can pick up each card and read it.

The binding constraint of the age was machine time. Tabulators were expensive and the work was arranged around them: claims waited for the daily run, and the run waited for the cards. Chapter 3 tells how such shops worked and why the design was sound for its limits.

2026: the same claim, one night

Seventy years later, a fire starts in a detached workshop just before midnight. When the fire department has left, the policyholder opens the insurer's web portal on a laptop and reports the claim. The portal does not hold the claim itself. It passes the report to the intake module, part of one deployable unit adopted in 2024 that also makes the initial reserve estimate, the first guess at what the claim will cost.

Intake checks the report against its published interface and gives it a reference. It hands the new claim to the core through the CICS entry transaction adopted in 1982, which stages it as an intake record for the next batch whose cutoff it meets; intake never writes the core's files itself. Then the triage agents start. They are AI agents, live since September 2025, and their job is narrow. They confirm that the policy is in force through one CICS transaction on the mainframe, the only same-day view of policy data, and they read its terms, such as limits and deductibles, from an event log fed from the DB2 copy of that data, which is a day behind. They propose a route for the claim: here, fire damage to an outbuilding, within the policy's cover, a candidate for a fast inspection. Each proposal goes through a policy check that enforces what the agent's tools are allowed to do, and each step leaves a record of what the agent read and why it proposed what it did. No agent approves a payment or writes the reserves the insurer reports. They propose; rules and people decide.

The family wants help with a hotel room. An intake rule, written by people and not by an agent, approves small advances below a set limit; anything larger goes to an adjuster. This advance is small and is approved. At this point the claim exists in the new layers, but not yet in the insurer's books.

That happens at night, on the mainframe, which an outsourcer has operated for the insurer since 2004. The claims core adopted in 1974 is a set of COBOL programs run in sequence by job control, with sort steps between them. It reads the day's intake records, updates the open-claim masters in IMS and the sequential and VSAM files beside them, and creates the claim of record: the version of the claim that the accounts, the regulators and the reinsurers rely on. In the same cycle it updates the case reserves, the adjusters' estimates of what each known claim will cost. At each month end the core also runs the recalculation of the reserve for claims incurred but not yet reported, an actuarial estimate for the whole book, not for

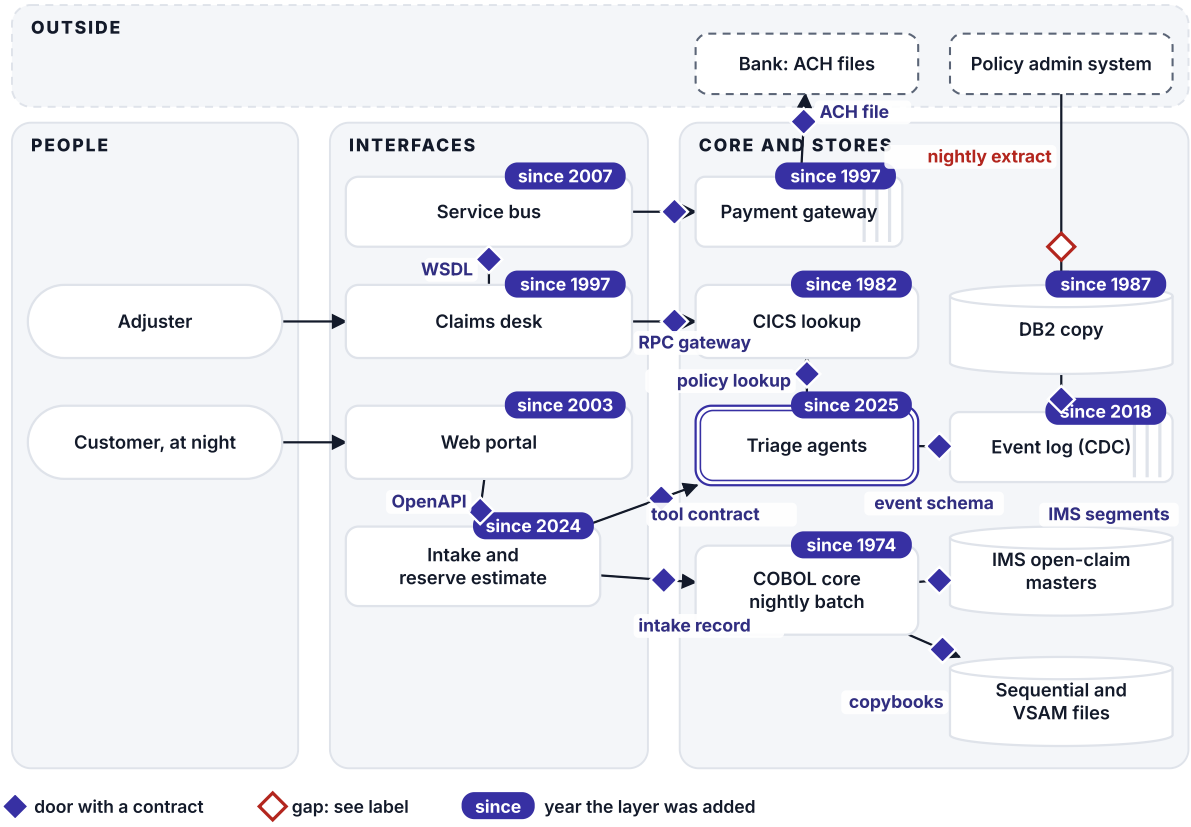
any one claim. Every program in that run reads its records through copybooks, shared descriptions of each record's layout that have been changed, carefully, for fifty years. Between 1997 and 1999 they were changed for the year 2000. A rewrite of the whole core was started in 2006 and abandoned in 2009. The batch still runs every night.

Before dawn, another job copies the day's policy changes from the separate policy administration system, the master of policy data, into DB2, the relational database the insurer added beside the core in 1987. Unlike every other crossing in Figure P.1, that nightly extract has no written specification that both sides check. Its layout lives in the extract program. Chapter 1 shows what that can cost.

Two days later an adjuster inspects the workshop. She works in the claims desk, a Windows application the insurer adopted in 1997, which reaches the mainframe's CICS transactions through a gateway program the insurer first wrote in 1991, so that branch PCs could call them. She agrees the amount. The payment instruction leaves the desk over the enterprise service bus, adopted in 2007, to a payment gateway that has sent payments as ACH files to the insurer's bank since 1997. The money reaches the family's account as an electronic transfer. No check is printed.

FIGURE P.1

The 2026 claim crosses layers the insurer added over half a century



A fire claim reported on the web portal one night in 2026 passes through layers the insurer adopted between 1974 and 2025; every crossing has a written, agreed description (an interface, a copybook, a tool definition) except the nightly extract of policy data into DB2.

Author's diagram of the invented insurer, drawn from its layer register (adoption years).

What the two claims share

Put the two journeys side by side and the differences are obvious: a claim that waited days for the daily card run against one routed within minutes of being reported, paper against screens, a plug board against software that writes its own reasons. What they share is less obvious and more useful.

Both claims cross a series of boundaries, and at each one the two sides rely on a shared description of what crosses. In 1956 that was the card layout: column positions every machine and person used. In 2026 it is the portal's interface description, the agents' tool definitions, the copybooks, the CICS transaction and the bus's service descriptions. The stronger ones let a team change its own side without asking everyone else, as long as it keeps the agreement. David Parnas set out that idea, information hiding, in 1972: put each

decision likely to change behind an interface that does not. A shared layout such as a copybook is the weaker form, since every program that uses it must change with it; Chapter 5 returns to the difference.

Both claims also depend on something that would show a failure. In 1956 it was the control total. In 2026 it is the evidence the agents leave, the reconciliation of the nightly run, and a bank statement that has to match the payments sent. Where a crossing has no agreement and nothing that would show an error, a fault can pass silently. Only one crossing in Figure P.1 is like that, and it is the nightly extract.

Finally, both systems are layered. The 2026 claim touches layers adopted in 1974, 1982, 1987, 1991, 1997, 2003, 2007, 2018, 2024 and 2025; others it does not touch, such as the 2013 broker API, still run beside them, and policy administration is a separate system outside this list. The book reads each one as an answer the insurer could afford to the problem that pressed hardest at the time, though Chapter 2 shows that vendors and fashion pushed too. Each later layer was built around the earlier ones instead of replacing them. That is accretion, and Figure 2.2 maps it layer by layer.

How this book is built

Part I gives the lens. Chapter 1 asks what architecture actually decides, and introduces the Decision Card. Chapter 2 sets out the binding constraint of each decade, tests the idea, and shows how to read an old design by the limits of its own time.

Part II takes the decades one at a time, from before 1960 to the 2020s. Each chapter tells the decade's main ideas, never more than eight landmarks, and shows the insurer's system in the same frame, so the reader watches it grow. Each one also tells one failure that stayed silent until it was too late: from Chapter 4 on, a real case from a primary source; in Chapter 3, the kind of error the card shop's own controls could not catch. Each ends with a worked Decision Card for a decision the insurer faced in that decade.

Part III asks what came back, what did not carry over, and what to do now with a system like this one: for each layer, keep it, wrap it, or replace it.

Monday rule: trace one real transaction through your own system end to end, and mark every crossing where nobody wrote down what passes.

PART I · THE LENS

CHAPTER ONE

What Architecture Decides



On a night in November 2025, every job in the insurer's claims system finished normally and every message matched its schema, yet over the following day new fire and water claims on home and commercial policies were routed on a wrong number. Nothing crashed. The fault entered at the one crossing in the system whose layout nobody had written down, and it was found the morning after by an adjuster who compared two screens. This chapter uses that night to ask what architecture is: not the diagram, but the set of decisions about where the boundaries are and what is agreed at each one.

A crossing that failed quietly

The insurer's policy data is mastered in a separate policy administration system. Each night a file of policy changes is copied into the claims side's DB2 tables by an extract program, and since 2018 every change to those tables has also been published as an event on a log. The triage agents, live since September 2025, read policy terms from that log. It is a day behind, but it is cheap to read and covers every policy; they call the same-day CICS lookup only to confirm that a policy is in force.

That November, the policy administration team widened the deductible to hold larger commercial amounts. To keep the record length and every other field where it was, they put a wider deductible field in the spare space at the end of each record and stopped filling the old one, which now held zero. To fill the new field, they sent every home and commercial policy in that night's file, not only the ones that had changed. They told the claims team, as they always had, by email, to people who did not maintain the extract; the gap was in process as much as in code. The extract program, written in 1987, kept reading the old position, so every one of those policies arrived in DB2 with a deductible of zero.

Nothing complained. The load ran long and the log carried far more events than usual, but nothing compared volume with a baseline. The load job ended with a normal return code. The record count in the file's trailer matched the rows loaded, because every record had

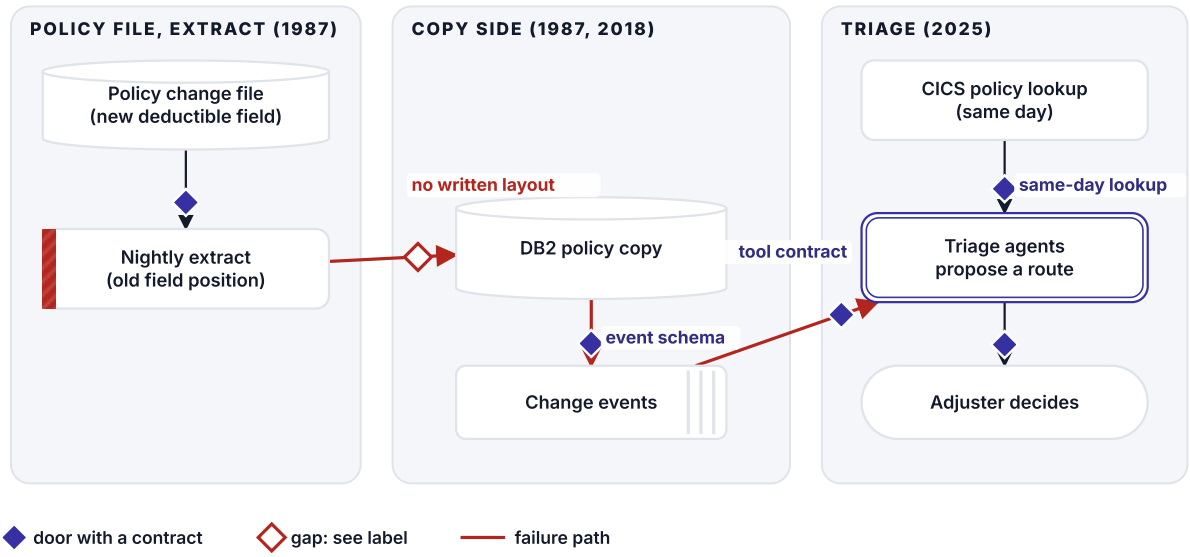
arrived; nobody totaled the deductibles. The change data capture published every altered row, and each event matched its schema, because a zero is a valid amount. That is the silent part: the fault crossed the extract and raised no error at any door it passed.

The agents did what their tools allowed. For claims reported on those policies after the load, through the day and the evening, they saw no deductible, so small losses looked payable in full, and they proposed fast inspection routes. The policy check that stands in front of each proposal confirmed that the route was one the agent may propose. It had no way to know the input was wrong.

The next morning an adjuster opened one of those claims at her desk. The CICS policy-lookup screen, which reads the policy the same day, showed a deductible of 2,500. The agent's record of what it had read showed zero, and where it had read it. With that record the team traced the value back to the extract in under an hour. No money was paid wrongly: the agents only propose, an adjuster decides, and the intake rule that approves small advances does not read the deductible. But for about a day the system had been confidently wrong, and every check had reported it as right. The repair, a change to the extract and a reload, was finished by the end of that working day.

FIGURE 1.1

The one crossing without a written layout is where the wrong value got in



In the book's November 2025 case every job ended normally; the wrong value entered at the nightly extract, the only crossing whose layout nobody had written down, and travelled on through doors that checked shape, not meaning.

Author's diagram of the invented insurer (an illustrative case inside this book).

Each check in that chain did its job. The schema checked shape, the policy check checked permission, the job scheduler checked return codes. The failure lived at a boundary that had never been given an agreed, checked description: the file layout between two teams. The decision that made it possible was taken in 1987, when someone chose to copy policy data into DB2 by a program that knew the layout privately. That choice was reasonable for its decade. It was also architecture, though nobody called it that.

What the definitions say

By the early 1990s, researchers set out to found software architecture as a field of study. Their definitions disagree on detail but agree on one thing: architecture is more than a picture of boxes.

Dewayne Perry and Alexander Wolf, in "Foundations for the Study of Software Architecture" (ACM SIGSOFT Software Engineering Notes, October 1992), proposed a model with three parts: "Software Architecture = {Elements, Form, Rationale}". Elements are processing, data or connecting elements. Form is the properties of the elements and their relationships. Rationale "captures the motivation for the choice of architectural style, the choice of elements, and the form." They also named two ways an architecture decays. "Architectural erosion is due to violations of the architecture." "Architectural drift is due to insensitivity about the architecture." In their terms, the 1987 extract was a small case of drift: by 2025 nobody remembered its private copy of the layout.

Philippe Kruchten, then at Rational Software, published "Architectural Blueprints: The '4+1' View Model of Software Architecture" in IEEE Software in November 1995. He began with the diagram problem: "Are the boxes representing running programs? Or chunks of source code? Or physical computers?" His answer was four views, logical, process, development and physical, each for different stakeholders, tied together by a fifth, a small set of scenarios. He was clear about what the views organize: the description of an architecture, which he glossed as "the decisions made".

The standards came last. IEEE 1471, an IEEE standard adopted in 2000, defined architecture as "the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution." ISO published it as an international standard, ISO/IEC 42010, in 2007, and the joint revision, ISO/IEC/IEEE 42010:2011, changed the wording to "fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution." A second edition, ISO/IEC/IEEE 42010:2022, replaced it in November 2022. Every version separates the architecture itself from the description of it. In the standard's terms, a diagram belongs to the description, not to the architecture.

Decisions, not diagrams

If architecture includes rationale and principles as well as elements, as these definitions say, then the reasoning is part of what has to be kept. That idea is older than the field.

In July 1970 Werner Kunz and Horst Rittel published "Issues as Elements of Information Systems" as Working Paper No. 131 of the Institute of Urban and Regional Development at the University of California, Berkeley. They described Issue-Based Information Systems, or IBIS, which "are meant to support coordination and planning of political decision processes." The elements of IBIS are "topics, issues, questions of fact, positions, arguments and model problems." Their subject was public planning, but the structure traveled: a question, the positions on it, and the arguments for and against each. In 1988 Conklin and Begeman described gIBIS, a graphical tool that implements the IBIS method. In 2005 two papers made the point for software in their titles: Tyree and Akerman's "Architecture Decisions: Demystifying Architecture" (IEEE Software, March 2005) and Jansen and Bosch's "Software Architecture as a Set of Architectural Design Decisions" (WICSA 2005).

On 15 November 2011 Michael Nygard published "Documenting Architecture Decisions" in a blog post. He described the inheritor's problem: "A new person coming on to a project may be perplexed, baffled, delighted, or infuriated by some past decision." Without the reasoning, that person has two choices, which he named "Blindly accept the decision" and "Blindly change it." His answer was a short text file for each decision, kept with the code, with five parts: Title, Context, Decision, Status and Consequences. A record could be proposed, accepted, deprecated or superseded by a later one. He called each one an architecture decision record, or ADR. The ThoughtWorks Technology Radar listed lightweight ADRs in its Trial ring in November 2016 and moved them to Adopt in November 2017.

The 1987 extract had no such record. Neither the person who changed the policy file in 2025 nor the team that built the triage agents could have read one. Each worked without the reasoning behind the 1987 choice. A real loss with the same shape, data crossing an interface whose units nobody checked, is the Mars Climate Orbiter in 1999, told in Chapter 7.

The Decision Card

This book's tool, the Decision Card, keeps Nygard's idea of one short record per decision, and adds fields that, in this book's illustration, would have helped on the November night. It names the binding constraint, so a reader can tell whether the decision is still right. It lists each boundary and what is agreed at it, so a changer knows what not to break. It says how the decision fails and what would show the failure. And it ends with a change guide, written for whoever changes it next, whether a person or an agent.

Extract of a card for the 1987 extract decision: eight of the ten fields, as a 1987 writer could have filled them in

Field	What the card would have said
Header	DC-1987-01; 1987; status Accepted; supersedes none; owner: head of applications development
Decision	Copy the day's policy changes from policy administration into DB2 each night with an extract program
Outcome and binding constraint	Policy reporting without touching the policy files; the constraint, as the department saw it, was the queue of applications waiting to be written
Boundaries and what crosses them	Policy file to extract: the layout is held in the program, not published; a record count in the trailer
Options rejected	Reports run against the policy files themselves (each new report waits in the queue); a layout agreed in writing with policy administration (the files belong to another department, with its own library and schedule, and every change would need both teams)
Quality, security, visibility, cost	Cheap to run; only the batch job writes the copy; nobody checks values against the source
How it fails and how we would see it	A layout change upstream puts wrong values in DB2; the job still ends normally; seen only by comparing with the source
Assumptions and the trigger to revisit	Only reports read the copy, and people check reports. Revisit if the copy is used for anything but reports
Change guide	Any change to the policy file layout must change the extract on the same day (a rule the claims side could not enforce on the policy team)

The last two rows matter most. The trigger fired in 2018, when the event log began to publish the DB2 copy, and again in 2025, when agents began to act on it. No record held that trigger, so no one was prompted to look. The full card is described in the Reader's Brief and

printed blank in Appendix A; each decade chapter fills one in.

JUAN'S LENS

I read that November night as a lesson about where to spend the effort. The team had invested in the new crossings: the tool definitions, the policy check, the evidence trail. All of them worked, and the evidence trail is why the fault was traced in under an hour once an adjuster spotted it, not at month end. The weak point was an old crossing that everyone had stopped seeing. When agents start to read a system, every quiet copy they read becomes a front door, and it needs a written, checked description like any other.

SECURITY LENS

Who can change it. The extract's input could be changed by anyone who changed the policy file, with no review on the claims side: an integrity gap, even with no attacker. The outsourcer that has run the mainframe since 2004 ran the job, but neither it nor the claims team owned the layout. **How fast it is fixed.** The fault was found the next morning and repaired that day because the evidence trail named the source; without it, the wrong value could have run until a monthly reconciliation. **What AI changes.** Agents read faster and wider than people, so a silent error in a copy reaches more decisions before anyone looks.

Monday rule: list every copy of data that an agent or a new service reads, and for each one name who owns its layout and what would show you it had changed.

CHAPTER TWO

The Binding Constraint, and How to Read Old Designs



In 1969, IBM began to price many of its programs separately from its machines. The Computer History Museum describes the move as IBM having "unbundled many programs (separating software from hardware)" and sums up the effect in one line: "Overnight, software changed from a giveaway to a competitive commercial product." Nothing technical ran out that day. A legal and commercial decision changed what companies could buy and what they had to build. This chapter sets out the book's main tool, the ladder of binding constraints, and is honest about its limits: some of the biggest changes in this history, like that one, came from somewhere else.

The ladder

The Reader's Brief gave the short form: in each decade, one resource ran out first at the leading edge, and the designs of that decade were answers to it. Figure 2.1 adds a second constraint to each rung, the one that pressed next, and an indicator where a primary source from the period measured something.

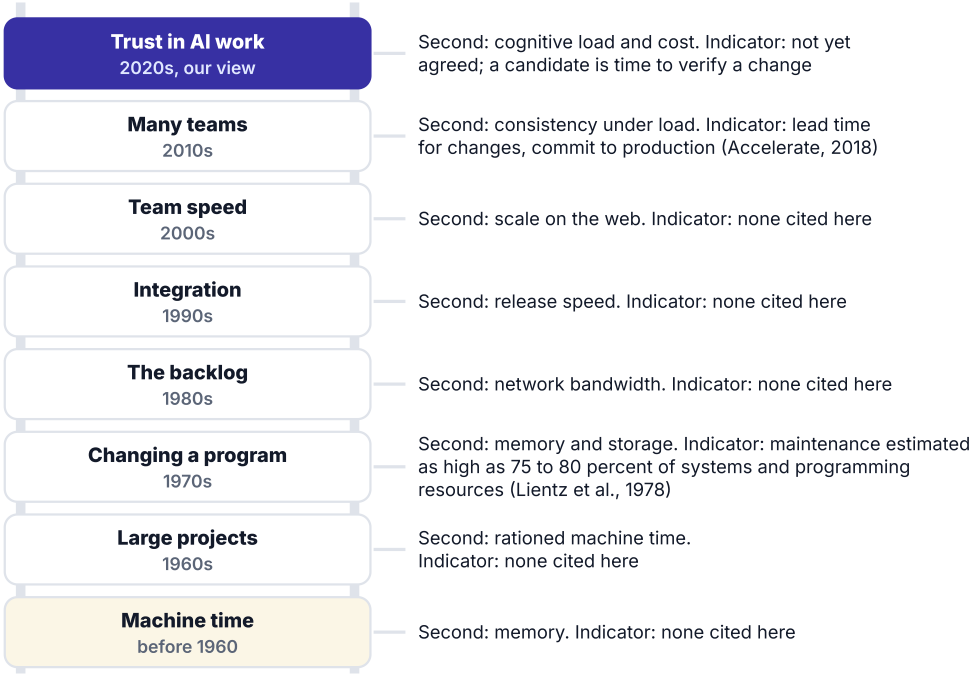
The second constraint matters because designs answer both. In the 1960s the binding problem was the complexity of large projects, but computers were still expensive and shared, so time-sharing systems, such as the one Corbató and his colleagues described in 1962, let many users work on one machine at once and see their results without waiting for a batch run. In the 2010s the binding problem was coordination between many teams, but data had to stay consistent under heavy load, so the services that let teams move alone also forced hard choices about consistency. Eric Brewer, who stated the trade in 2000 as a choice of at most two of consistency, availability and tolerance to network partitions, gave the corrected form in 2012: designers "need to choose between consistency and availability when partitions are present", but "because partitions are rare, there is little reason to forfeit C or A when the system is not partitioned." In his words, "the '2 of 3' formulation was always misleading." Heavy load alone is not that condition; a network partition is.

Indicators are harder. A constraint is easy to name after the fact and hard to measure at the time. Two rungs have an indicator from a primary source of their own time, and one earlier source gives context. In "Software and Its Impact: A Quantitative Assessment", a RAND paper dated December 1972 and written for *Datamation*, Barry Boehm estimated that "Overall software costs in the U.S. are probably over $10 billion per year", and concluded that overall system cost is generally minimized "by procuring computer hardware with at least 50 percent to 100 percent more capacity than is absolutely necessary." This book reads that advice, to buy spare hardware to save programmer time, as evidence that machine time had stopped being what ran out first at the leading edge; it is context for the ladder, not a rung indicator. For the 1970s rung itself, the cost of changing programs, Lientz, Swanson and Tompkins surveyed 120 organizations, of which 69 replied (Communications of the ACM, June 1978): maintenance and enhancement "consume a major portion of the total life cycle cost of a system", and "Rough estimates of the total systems and programming resources consumed range as high as 75-80 percent in each category." In 2018 Nicole Forsgren, Jez Humble and Gene Kim published *Accelerate*, built on the State of DevOps survey research, which measured delivery with four key metrics: deployment frequency, lead time for changes, time to restore service, and change failure rate. DORA's guide, updated in January 2026, now lists five metrics, with new names for some of the original four. Lead time, the time from a change being committed to it running in production, is a thin proxy for the 2010s rung: waiting on other teams makes it long, though much of that waiting happens before the commit.

For the other rungs this book gives no number rather than an invented one. The 1980s backlog of applications that users waited for was surveyed at the time, but this book cites no single measure of it. For the 2020s, the indicator is still being argued; Chapter 10 offers a view.

FIGURE 2.1

Each rung with its second constraint, and an indicator where a primary source gives one



Two rungs carry a sourced indicator from their own era; the others are left without one rather than given a number nobody measured.

Author's synthesis. Indicators: Lientz, Swanson and Tompkins, CACM (June 1978); Forsgren, Humble and Kim, Accelerate (2018).

The ladder applies Goldratt's logic, set out in the Reader's Brief, to an industry over decades, not to a plant: once one bottleneck is relieved, another binds.

Drivers that are not constraints

Not everything that shaped architecture was a scarce resource. Four other forces appear throughout the book, and a reader should keep them apart from the ladder.

- **Regulation.** Laws on financial reporting, privacy and operational resilience changed what systems had to record and prove. They did not make anything scarce; they made some failures unaffordable. The decade chapters give the laws and their dates.
- **Buy or build.** Independent products such as ADR's Autoflow already existed, but after IBM unbundled many programs in 1969, buying a program, and later a whole package, became a common choice. An insurer's choice between a bought package and its own code is often settled by cost, risk and staff, not by the leading-edge constraint.

- **Vendor push.** Vendors set the pace of what was available, and support ended for products whether or not customers were ready.
- **Fashion.** Some ideas spread because others were adopting them; such a design cannot be judged by a constraint it was never meant to answer.

The insurer felt all four. Its rewrite of the core, started in 2006 and abandoned in 2009, shows what happens when such pressures meet the cost of replacing a system that works. Chapter 8 tells it.

Leading edge and installed base

The ladder describes the leading edge: the firms and laboratories that met each constraint first. Most companies met it later, and kept what they had. Figure 2.2 is the insurer's accretion map. Counting a layer as a line of software and its interfaces, not the machines under it, which have been replaced many times, only two layers have been switched off since 1956: the card shop, retired in 1962, and the 1401 programs, retired with the 1974 core. This counts the 1967 emulation together with the programs it ran; Chapter 12 lists their retirements separately. The service bus is partly retired. Every other layer still runs, wrapped by the layers that came after it. The first wrap came in 1967, when the insurer moved to System/360 and kept its 1401 programs running unchanged under emulation, which IBM offered as a compatibility feature. The pattern repeated for sixty years.

FIGURE 2.2

The accretion map: each layer of the claims system is born, wrapped or retired, and most are still running



Of the layers the insurer added since 1956, only the card shop (1962) and the 1401 programs (1974) have been switched off, and the service bus is partly retired; the rest still run in 2026, wrapped by later layers.

Author's diagram of the invented insurer; all years are the insurer's adoption years.

Figure 2.3 shows the gap between release and adoption. IBM introduced the 1401 on 5 October 1959; the insurer moved its claims run to a 1401-class machine in 1962. IBM announced IMS for its mainframes, starting with System/360, in 1968; the 1974 core keeps its open claims in it. IBM first shipped CICS, as PU-CICS, in 1968, and the CICS program product followed on 8 July 1969; the insurer put its claims inquiry on CICS in 1982. IBM announced DB2 on 7 June 1983 and launched version 1.1 in 1985; the insurer added DB2 in

1987. IBM released MQSeries in 1993; the insurer's payment queue came in 1997. The CORBA 2.0 specification, including IIOP, is dated July 1995 and was updated in July 1996. The OMG's history records its CORBA 2.0 milestone in August 1996, the date plotted here; the insurer's claims desk used it in 1997. CERN put the web's software in the public domain on 30 April 1993; the insurer's portal arrived in 2003. Docker was first shown in public in March 2013; the insurer's first containers came in 2016.

FIGURE 2.3

Leading edge and installed base: the year a technology appeared is not the year the insurer adopted it



For each layer, the first dot is the vendor's or standards body's date (an announcement for the 1401, System/360 and IMS, a first shipment for CICS, general availability for DB2, a first public showing for Docker, the OMG's August 1996 history milestone for CORBA 2.0, a release for the rest) and the second is the insurer's adoption year; the gap runs from about one year to fourteen. CORBA's specification is dated July 1995 and updated July 1996; the plotted milestone is not its first specification date.

Release dates: IBM history pages and announcements, IBM Redbooks, OMG CORBA history, CERN, Docker. Adoption years: the invented insurer's layer register.

Counted from those dates, which mix announcements and first releases as the figure notes, the lags run from one year to fourteen (thirteen for CICS if counted from the program product). The book reads them this way: a mid-size insurer adopted a technology when staff could run it and when the old system had become more expensive to keep than the new one

to build. A new layer at the edge, like the 1997 object broker, could come quickly; the core moved slowly. That is the installed base's own binding constraint: the cost and risk of change, set against a system that already works.

So there are two questions to ask of an old design, and they have different answers. What constraint did the leading edge face when the idea appeared? And what constraint did this company face when it adopted it? The 1974 claims core was a sound answer to the second question in 1974. Whether it is still sound depends on which constraint binds the insurer now, not on the age of its code.

What would prove the frame wrong

The ladder would be wrong if a decade's most lasting designs spent freely the very resource the ladder says was scarce, or if the same design ideas appeared regardless of what was scarce. Three of the most important ideas in this history fit it badly, and they are worth naming.

The relational model. Edgar Codd's "A Relational Model of Data for Large Shared Data Banks" appeared in June 1970. Its first sentence states a practical aim: "Future users of large data banks must be protected from having to know how the data is organized in the machine." That is the cost of changing programs, the 1970s rung, named at the start of the decade. But the answer came from mathematics, the theory of relations. IBM's own history dates the System R project, an industrial-strength implementation, to 1973, yet IBM announced DB2, its relational database for its MVS mainframes, only in 1983. The model fits the ladder in its motive; what set its timing lies outside the ladder.

Unix. Dennis Ritchie's account dates Unix to 1969, when Bell Labs had withdrawn from the Multics project and Ken Thompson found "a little-used PDP-7". Scarce access to computing played a part. But Ritchie names the group's aim as social: "a system around which a fellowship could form."

The web. Tim Berners-Lee wrote his first proposal at CERN in March 1989, under the title "Information Management: A Proposal". It answered a laboratory's problem, not the decade's binding constraint, and CERN put the software in the public domain four years later.

None of the three spent a scarce resource freely, so the first test is not failed. The second test is harder. Time-sharing in the 1960s and cloud computing in the 2010s share one idea, many users on one pooled machine, under different binding constraints; Part III reads such returns as an old idea answering a new constraint, which is a reading, not a proof. What the

three also show is that invention has other causes. On this evidence the ladder explains why ideas were adopted widely, and when, better than why they were invented. Read it as a tool for judging designs, not as a law of history.

Constraints stack, and borders are arbitrary

A constraint does not vanish when a new one binds. It moves down the ladder and waits. Machine time was the binding constraint before 1960 and stopped being one by the 1970s. It returned in the 2010s as the monthly bill for cloud computing, and again in the 2020s as the cost and supply of the processors that run AI models. A design that ignores a constraint because it no longer binds can be caught out when it returns.

The decades are also a convenience. The 1401 was introduced in 1959 and shaped practice in the 1960s; the web was proposed in 1989 and changed the 1990s. Where an idea crosses a border, the book puts it in the decade it shaped.

> **OUR VIEW, AS OF NOVEMBER 2026**
>
> The rung for the 2020s is a judgment, not a measurement: trust in work that AI made, and the cost of checking it. If that is right, the scarce thing in a design review is no longer the time to write code but the time to verify it, and designs that make verification cheap will win, as designs that saved machine time once did.

> **SECURITY LENS**
>
> **Who can change it.** Each rung widened the set of people who could change a system: operators, then programmers at terminals, users with PCs, many teams, and now agents. **How fast it is fixed.** Adoption lag says nothing by itself about how fast a fix ships; measure that separately, as the time from a published fix to the fix running in production. **What AI changes.** Agents can write a change in minutes, so the question of who checks a change before it ships becomes the main security question.

Monday rule: for the oldest system you run, write down two years, when its main technology appeared and when you adopted it, and the constraint that bound you at the second date.

CHAPTER THREE

Before 1960: Wiring the Machine



On 6 January 1954, Hans Peter Luhn of IBM filed a patent application for a "Computer for verifying numbers". Its stated object was "a simple, inexpensive and portable computer for computing check digits" and "a simple device for verifying numbers which have a check digit appended". The patent, US 2,950,048, was granted on 23 August 1960. Luhn was working on a problem every punched-card shop knew: a number copied by hand from a form onto a card could be wrong, and nothing in the number itself would say so. This chapter is about the machines and the habits that grew up around that problem, and about one decision the insurer took in 1956: to put its claims on cards, in batches, and to release no batch until its totals agreed.

Before 1960 the binding constraint was the machine itself. Electronic computers were few, slow, expensive and hard to program. Punched-card machines were plentiful, but each report had to be wired by hand. The scarce thing was not ideas about software; it was machine time and the people who could make a machine do one more job. Designs from this era make sense once you see them that way.

The program in the wiring

Punched cards were already old by the 1950s. The US Census Office leased Herman Hollerith's tabulating machines for the 1890 census, and the Census Bureau records that the count "finished months ahead of schedule". By mid-century, a business data shop was a room of unit-record machines: key punches and verifiers to make the cards, sorters and collators to order and merge them, and an accounting machine to add them up and print. IBM's first manual for its 407 accounting machine is dated 1949: the machine read punched cards, added them up and printed the results, and what it printed was set by a wired control panel.

The program of such a shop lived in two places. The first was the wired control panel: a plugboard on which an operator connected card columns to counters and print positions with short wires. Each report had its own panel. Change the report and you rewired the panel, or swapped in another one. The second was the card layout: which columns held the claim number, which held the date, which held the amount. Every panel, every form and every clerk had to agree on that layout. It was an agreement shared by parts that changed at different speeds.

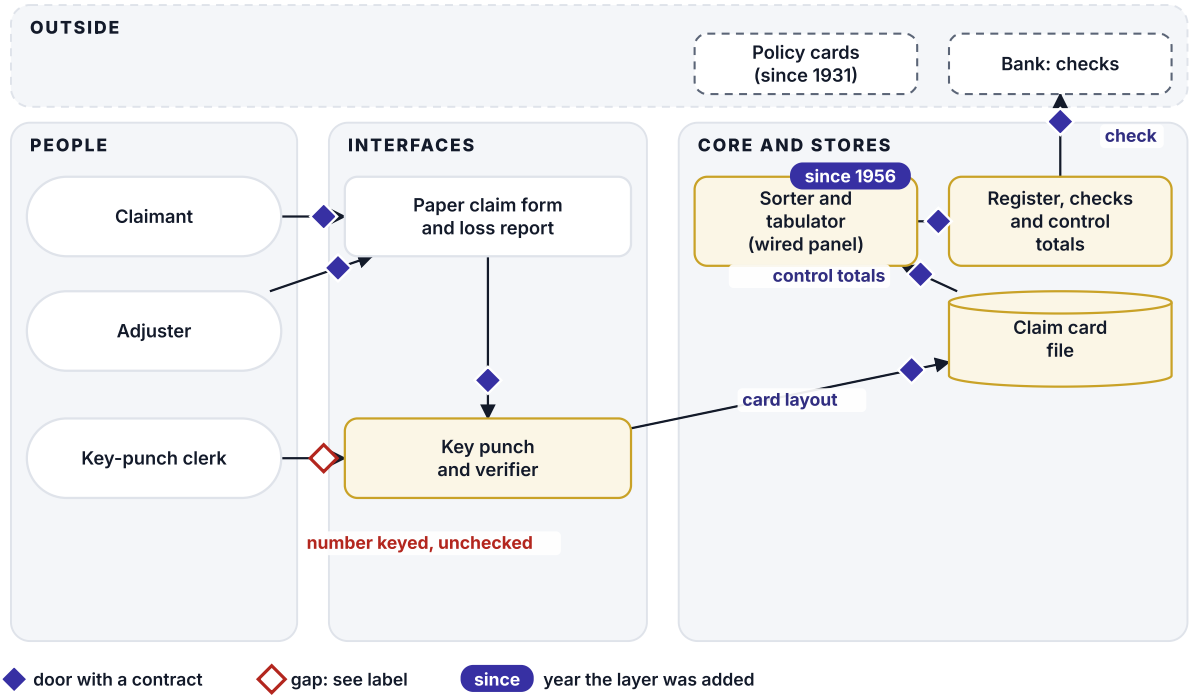
The insurer in this book already kept its policy records on punched cards from 1931, as told in *From Tally to Token*. Claims tabulation followed in 1956. A claim arrived as a paper form and an adjuster's loss report. A clerk punched it, a second clerk verified it by keying it again, and the cards went in batches, each with a ticket carrying its count of forms and its total of amounts, through the sorter and the tabulator. The tabulator printed the claims register and the totals, and the shop prepared checks from them.

THE CLAIMS SYSTEM · 1956 · PUNCHED-CARD CLAIMS TABULATION

What was added	Claims are punched on cards and tabulated in a unit-record shop, as US insurers such as Metropolitan Life and Prudential did.
Door and its contract	The card layout: which columns hold the claim number, the amount and the date
In 2026	Retired; the card layouts survive as field positions in the core's files

FIGURE 3.1

1956: the card shop. The card layout is the shared contract; the keyed number has no check beyond re-keying



The insurer's 1956 claims tabulation on the fixed canvas every decade figure reuses: people on the left, interfaces in the middle, the core and its stores on the right, the outside along the top.

Author's diagram of the invented insurer, based on IBM Type 405 Principles of Operation (Computer History Museum 102698809) and IBM 407 control panel manual (Computer History Museum 102724934).

Figure 3.1 sets the canvas that every decade figure in this book reuses. The card layout already carries a marker: it is the agreement every machine in the shop reads. The keyed number does not. Beyond keying it a second time, nothing in the 1956 shop could tell a right claim number from a wrong one that was punched consistently.

The stored program and the subroutine

While insurers wired panels, a different idea was taking hold in laboratories. John von Neumann's *First Draft of a Report on the EDVAC*, dated 30 June 1945 and circulated by the Moore School of Electrical Engineering under his name alone, described a machine that would hold its instructions in the same memory as its data. A program would be stored, not wired. It could be loaded, changed and replaced without touching the hardware.

At Cambridge, EDSAC ran its first logged program on 6 May 1949: it computed squares of the numbers 0 to 99. The laboratory soon faced the question every programmer since has faced: how do you avoid writing the same instructions twice? Its answer was the closed subroutine, credited to David Wheeler. A routine for one task, such as a square root, lived in a library. A program jumped to it, the subroutine did its work and returned to the place it was called from. The jump became known as the "Wheeler jump". In 1951, Wilkes, Wheeler and Gill published *The Preparation of Programs for an Electronic Digital Computer* (Addison-Wesley), which described the EDSAC library of subroutines. Wheeler presented "The use of sub-routines in programmes" at the ACM national meeting in Pittsburgh in May 1952.

The subroutine put a boundary inside the program itself. Its caller did not need to know how the square root was computed; it needed to know where to put the argument, where to find the answer and how control came back. That calling convention was an agreement in the same sense as the card layout. It returns, decades later, as the remote procedure call, the object interface and the service API. Chapter 11 follows that line, and what did not carry over once the call crossed a network.

The stored program also reached business early. In Britain, the catering firm J. Lyons and Co. built its own machine, LEO I. Its Bakery Valuations job was run on real data as a test on 5 September 1951. The Centre for Computing History, citing the notebooks of LEO's Ernest Lenaerts, records the first regular run in three parts, the first finished on Wednesday 28 November 1951 and the last at 2:35 p.m. on Friday 30 November. The LEO Computers Society calls it "the first commercial data processing job in the world".

Insurers look at computers

Life insurers studied electronic computers early. Their problem was scale: millions of small policies, each with premiums, dividends and records to keep for decades. In September 1952 a committee of the Society of Actuaries issued its *Report of Committee on New Recording Means and Computing Devices*. M. E. Davis, vice-president and chief actuary of the Metropolitan Life Insurance Company, described the work at the Eastern Joint Computer Conference in December 1953. The plan that came out of it, he said, "has become known as the Consolidated Functions Plan".

Davis was careful about the medium. "We feel that insurance procedures intended for use in the next few years should not depend upon magnetic tape as a primary record keeping medium," he told the conference. His reasons were practical. Nobody knew "how long it would take for policy records in magnetic tape form to be acceptable to the state

departments" that supervised insurers, or when courts would accept tape as evidence, and the saving expected from tape over cards was small. The plan kept the policy file on punched cards.

The machines arrived soon after. Metropolitan Life's own history says that "in 1954, Metropolitan Life installed UNIVAC, the first large-scale electronic data processing system in the life insurance industry". IBM announced development of its 650 Magnetic Drum Data Processing Machine on 14 July 1953. IBM records that "John Hancock Mutual Life Insurance Company took delivery of the first IBM 650 in December 1954", and that "IBM eventually installed nearly 2,000 of the devices around the world". The 650 was a stored-program machine that a mid-size business could rent, and it read the same punched cards as the shop next door.

The insurer in this book did not buy one in the 1950s. Its card shop already worked, and in 1956 it chose to extend the shop to claims rather than replace it. Judged by its own constraint, the cost of machine time against the cost of clerks, that was a defensible choice for a mid-size firm; the large life offices that bought first had far more records to process.

Languages for the business

Stored programs had a cost of their own: they were written in the instructions of one machine. A program for one computer would not run on the next. Two efforts in the late 1950s began to move the program up from the machine.

FORTRAN came first, for scientific work. John Backus and his IBM team presented "The FORTRAN Automatic Coding System" at the Western Joint Computer Conference in February 1957. In his own history of the project, Backus wrote that "sometime in early April 1957 we felt the system was sufficiently bug-free to distribute to all 704 installations". A formula written once could now be translated for the machine by the machine, at first only for IBM's 704.

Business followed. On 28 and 29 May 1959, a meeting at the Pentagon considered "both the desirability and the feasibility of establishing a common language" for business data processing. Its work went on in the Conference on Data Systems Languages, CODASYL, and the specification of a Common Business Oriented Language, COBOL, was published by the Government Printing Office in April 1960. The insurer did not adopt COBOL for its claims core until 1974 (Chapter 5).

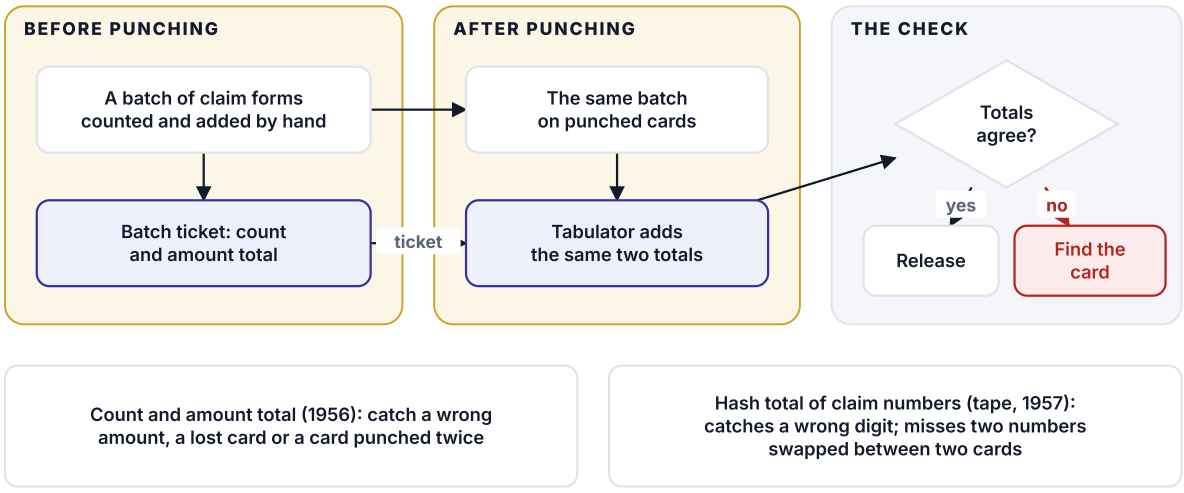
The promise of a common language was a promise about a boundary. If two machines from two makers accepted the same language, a program could cross from one to the other, and a business could change suppliers without rewriting its work. That promise was never complete.

Batch, totals and the operator

The habit that best defines this era is the control total. Machines and clerks both made mistakes; the defense was arithmetic done twice. Before a batch of forms went to the punch room, a clerk counted the forms and added the amounts on an adding machine. The total went on a batch ticket. When the cards went through the tabulator, the machine added the same amounts. IBM's operating manual for its 405 accounting machine describes a control that "permits the last card to feed through the machine and control totals to print after the last card". If the two totals agreed, the batch was released. If they did not, someone found the missing, doubled or mis-punched card before anything was paid. Agreement showed that the batch was complete and that its amounts added up; it did not show that every card was right. Two errors that cancel, such as one amount punched 10.00 too high and another 10.00 too low, still balance both the count and the total.

FIGURE 3.2

An amount total catches a lost or doubled card, not a wrong number



The insurer's 1956 batch is released when the count and amount total on its ticket agree with the tabulator's; a mis-keyed claim number passes, and so do two errors that cancel. A hash total of the numbers would catch a keying slip.

Author's diagram based on IBM Type 405 Principles of Operation (Computer History Museum 102698809) and IBM 705 Generalized Merge Program, form 32-7626 (1957).

A total of amounts has a blind spot, and Figure 3.2 shows it. If a clerk punches the right amount against the wrong claim number, the amounts still balance. The era's answer was a hash total: a sum of the claim or policy numbers themselves, meaningless as a number but sensitive to a wrong or transposed digit. IBM's 1957 manual for its 705 merge program

describes "a hash total accumulated from the first ten digits of the record" and a message at the end of the run: "Hash Totals Agree" or "Hash Totals Do Not Agree". That 1957 manual is the evidence cited here; the insurer's 1956 batch tickets carried only a count and an amount total. A hash total has its own blind spot: two numbers swapped between two cards leave the sum unchanged.

Around the totals stood the operator. A batch shop ran on schedules, run sheets and the person at the machine who knew which panel went with which job. Books for managers began to describe the whole system: Richard G. Canning's *Electronic Data Processing for Business and Industry* was published by John Wiley and Sons in 1956. In such a shop the design lay in the procedure: forms, cards, batches, totals and people, in that order, with a check at each hand-off.

THE ERA'S SILENT FAILURE

The mis-keyed number. Unlike the cases in later chapters, this is a mechanism, not a dated incident: the error the era's verifiers, totals and check digits were built against. A claim number or policy number copied from a form onto a card could be wrong in a way no total of amounts would show. A keying slip, such as two digits transposed, was usually caught when a second clerk keyed the card again. A misreading was not: when the form was badly written, both clerks could read a digit the same wrong way, and the verifier passed the card. The batch balanced, because its ticket held a count and a total of amounts and nothing about the numbers. When the swapped number was itself a valid policy number, the claim cards still matched a policy card, so the shop raised no error. The payment went against the wrong policy, and the first sign was a letter from a claimant or an auditor's sample months later.

The era began to build two defenses. The hash total tested a batch of numbers against a total taken from the source; but a clerk who added the numbers from the same badly written form would carry the same misreading into the total, so it caught keying slips, not misreadings. The check digit tested each number by itself: Luhn's patent, filed in 1954 and granted in 1960, appended a digit computed from the others, doubling every other digit, so that a machine or a clerk could verify the number on its own. Because the digit was computed when the number was issued, a misread number usually failed the check; a few swaps, such as 09 for 90, still passed. In 1956 the insurer had neither. The silent part was the number itself: it crossed from the clerk to the card with nothing that could tell right from wrong.

JUAN'S LENS

I read the batch ticket as the oldest proof I know in data processing. The clerk did not say the batch was complete; the clerk wrote down a number, and the machine had to produce the same one. Most systems I meet today assert that a feed arrived. Few recompute anything on the other side. For all the blind spots this chapter shows, the 1956 shop made the machine produce the clerk's number before a check went out.

SECURITY LENS

Who can change it. In a card shop, whoever held the panels and the card decks could change what the shop did. Control rested on physical custody: the machine-room supervisor wired the panels, and checks were signed outside the shop. **How fast it is fixed.** A wrong panel was rewired overnight; a wrong card was found by its batch and punched again. The fix was fast once the total disagreed and slow when nothing disagreed. **What AI changes.** An agent that edits a record layout today does what rewiring a panel did, for every reader of the layout at once. A count and a total recomputed by the receiver would show records lost or altered by such a change; they would not show that a field now means something else.

DECISION CARD · DC-1956-01

1956 **Retired** Owner: Claims office manager Supersedes: none (hand-posted ledgers)

1 Decision	Punch every claim on cards in a fixed layout and tabulate claims in batches, each released only when its control totals agree.
2 Outcome and binding constraint	Claims are registered, totaled and paid by check with figures the auditors can add up again. **Binding constraint:** Machine cost: a stored-program computer costs more per claim than rented card machines, while hand posting can no longer keep up.
3 Boundaries and doors	**Claim form to card**: the card layout: fixed columns for number, date and amount **Card deck to tabulator**: the batch ticket with count and amount total **Tabulator to register and checks**: the wired panel for each report
4 Options rejected	**Keep posting by hand**: volume and errors grow with the book of business **Rent a stored-program computer now**: its rental per claim is higher than a card shop that already works, and programmers are scarce
5 Quality · Security · Visibility · Cost	**Quality.** A batch is not released until its totals agree. **Security.** Only the machine-room supervisor rewires panels; checks are signed outside the shop. **Visibility.** The batch ticket and the printed totals on every run. **Cost.** Machine rental, cards, and a key-punch and verifier staff.
6 How it fails, and how we would see it	A clerk keys a wrong claim or policy number; the amounts still balance, so the batch passes. **Evidence:** Nothing in the batch: no hash total of the numbers; a claim card with a wrong but valid policy number still matches. A complaint from the claimant or an auditor's sample shows it.
7 Run	Shift supervisors in the machine room; a failed batch is punched again from the forms.
8 Reversal cost	Low in machines, high in habits: the layouts and totals outlive the machines.
9 Assumptions and Revisit when	Claims arrive on paper; payment within days is acceptable. **Revisit when:** When a stored-program machine costs less per claim than the shop.
10 Change guide	1. Never move a field to other columns without changing every panel and form that reads it. 2. No batch leaves the shop without agreeing totals.

Illustrative card for the invented insurer. Retired in 1962, when DC-1962-01 moved the claims run off the card shop; its layouts and totals live on.

Monday rule: pick one feed that enters your core and make the receiver recompute a count and a hash total before it accepts the batch.

CHAPTER FOUR

The 1960s: The Terminal and the Crisis



On 22 July 1962, an Atlas-Agena B rocket carrying the Mariner 1 probe to Venus lifted off from Cape Canaveral. NASA's space science data archive says the Range Safety Officer destroyed it 293 seconds later, "when it veered off course". Two faults met in flight. A beacon on the rocket failed, and the ground guidance program, in the mode it then fell back on, was missing a symbol that had been dropped when handwritten equations were turned into computer instructions. That error had sat in the program, unseen, until the beacon failed. This chapter is about a decade in which programs grew beyond what one person could hold in their head, and about the decision the insurer took in 1967: to change its machine without rewriting its programs.

In the 1960s the constraint shifted. Machines became cheaper, a business could rent one, and from 1964 IBM's new machines were compatible with each other. The scarce resource became the ability to build and change large programs: the complexity of big projects, the supply of programmers, and the cost of rewriting everything whenever the machine changed. By 1968 a conference of experts had a name for the problem. The decade's best ideas were answers to it.

The business computer: the insurer leaves the card shop

IBM introduced the 1401 Data Processing System on 5 October 1959. IBM records that "more than 5,200 orders arrived in the first five weeks" and that "by the mid-1960s, more than 10,000 of the 1401 systems were installed". A recollection published by the Computer History Museum's 1401 team puts the first shipment at the end of September 1960, to Time-Life in Chicago. It was a stored-program computer built for the work card shops already did: read cards, compute, print reports. At least one insurer used it: a 1965 photograph in the Computer History Museum's archive shows a 1401 installed at The Travelers.

The insurer moved its claims run to a 1401-class machine in 1962, about three years after the announcement and two after the first delivery. It waited until the machine was proven, the staff trained and the card shop's cost clearly higher. The new run kept the old card layouts as

its input. The forms, the key punches and the control totals stayed the same; what changed was that a program, not a wired panel, sorted, matched, computed and printed. A claim master now lived on magnetic tape.

THE CLAIMS SYSTEM · 1962 · STORED-PROGRAM BATCH RUN

What was added	The claims run moves from the punched-card shop to a stored-program batch run on an IBM 1401-class machine.
Door and its contract	The card and tape record layouts the programs read
In 2026	Retired; its programs ran on under emulation from 1967 until the 1974 core

DECISION CARD · DC-1962-01

1962 **Retired** Owner: Data processing manager Supersedes: DC-1956-01

1 Decision	Move the claims run from the card shop to a stored-program batch computer, keeping the card layouts as its input.
2 Outcome and binding constraint	The whole claims run, from sorting to register and checks, in one programmed pass each night. **Binding constraint:** Machine time and memory: a few thousand characters of storage on a rented machine.
3 Boundaries and doors	**Cards to program**: the 1956 card layout, unchanged **Program to tape**: the claim master record layout **Run to register**: the same control totals as the shop
4 Options rejected	**Expand the card shop**: more machines and staff for each new line of business **Rent an older drum-memory computer**: cheaper now, but slower with cards and printing, and already being replaced
5 Quality · Security · Visibility · Cost	**Quality.** Totals must agree with the card-shop totals during the parallel run. **Security.** Program decks are kept by the programming group; the operator runs them and never edits them. **Visibility.** Console log and printed control totals after each run. **Cost.** Machine rental; programmers, and programs written in a symbolic language tied to this one machine's instructions.
6 How it fails, and how we would see it	A program reads a field one column off after a form change and posts the wrong amount. **Evidence:** Totals that do not agree with the batch ticket.
7 Run	Operators on shifts; a failed run restarts from the last good tape.
8 Reversal cost	Medium: the programs are written for one machine's instruction set.
9 Assumptions and Revisit when	This machine, or one that runs its instructions, stays available for years. **Revisit when:** When the supplier announces a machine family that does not run these instructions.
10 Change guide	1. Keep the card layout stable: the programs and the shop share it. 2. Every run prints its totals before any check is released.

Illustrative card for the invented insurer. Superseded in 1967 by DC-1967-01; retired in 1974, when the core replaced its programs.

The 1956 card layout became the input record of the 1962 program. The decision to keep it saved the claims office from retraining and the program from a conversion of every card. It also meant the layout could no longer change without changing the program. From 1962 the layout belonged to two systems, the shop and the program, and it could change only when both did.

The terminal: sharing one machine among many people

While business ran in batches, universities asked a different question: could many people use one computer at once, each at a terminal, as if it were theirs? At MIT, Fernando Corbató's group demonstrated the Compatible Time-Sharing System (CTSS) in November

1961, and described it in "An Experimental Time-Sharing System" at the Spring Joint Computer Conference in May 1962. At Dartmouth College, at 4 a.m. on 1 May 1964, John Kemeny and a student programmer typed RUN at neighboring terminals of the college's new time-sharing system, running the first programs in BASIC. Kemeny, Thomas Kurtz and a group of undergraduates had built the system and the language together. A student at a teletype could write a program, run it and see the answer in seconds, not the next morning.

Business found its own use for terminals. American Airlines and IBM built SABRE, a reservation system that kept every seat on every flight in one place and let agents query it from terminals. IBM's history traces the project to a 1953 meeting on a flight and says American committed "roughly USD 40 million" to the joint effort. The Computer History Museum records that SABRE was "first tested in 1960" and "took over American's reservations four years later", linking "2,000 terminals in 65 cities to a pair of IBM 7090 computers". By the mid-1960s, IBM says, it was "handling 7,500 reservations an hour".

With SABRE, a terminal changed the shape of a system. A batch run could be checked after it finished; an online system had to be right while the customer waited. It needed one current record of each seat, rules about who could change it, and a way to survive a failure in the middle of a transaction. Those are the same questions the insurer would face in 1982, when it put claims inquiry online (Chapter 6). For the insurer of the 1960s, terminals were too costly for claims work. The constraint that kept its claims in batch was the price of the machine time an online system would need.

One architecture, many machines, and the first wrap

On 7 April 1964, IBM announced System/360: a family of computers, from small to large, that would all run the same programs. Gene Amdahl, Gerrit Blaauw and Fred Brooks explained the idea in the *IBM Journal of Research and Development* that month, in a paper that defined the system's architecture as "the attributes of a system as seen by the programmer, i.e., the conceptual structure and functional behavior, as distinct from the organization of the data flow and controls, the logical design, and the physical implementation". The architecture was a boundary between the programmer and the machine. Below it, IBM could build faster or cheaper models; above it, a program written once would run on any of them, given enough storage, the devices it used and the optional features it needed (Chapter 5 quotes the conditions).

The family had a problem: it did not run the programs customers already had. IBM's answer was emulation. With a special feature, a small System/360 Model 30 could execute 1401 instructions as well as its own; IBM's manual for the Model 30 describes a "1401, 1440, 1460

Compatibility" feature. An IBM history of the period says the company could bring out the new family "with emulators that could run existing customer programs". An old program ran unchanged on new hardware, behind a layer that imitated the old machine.

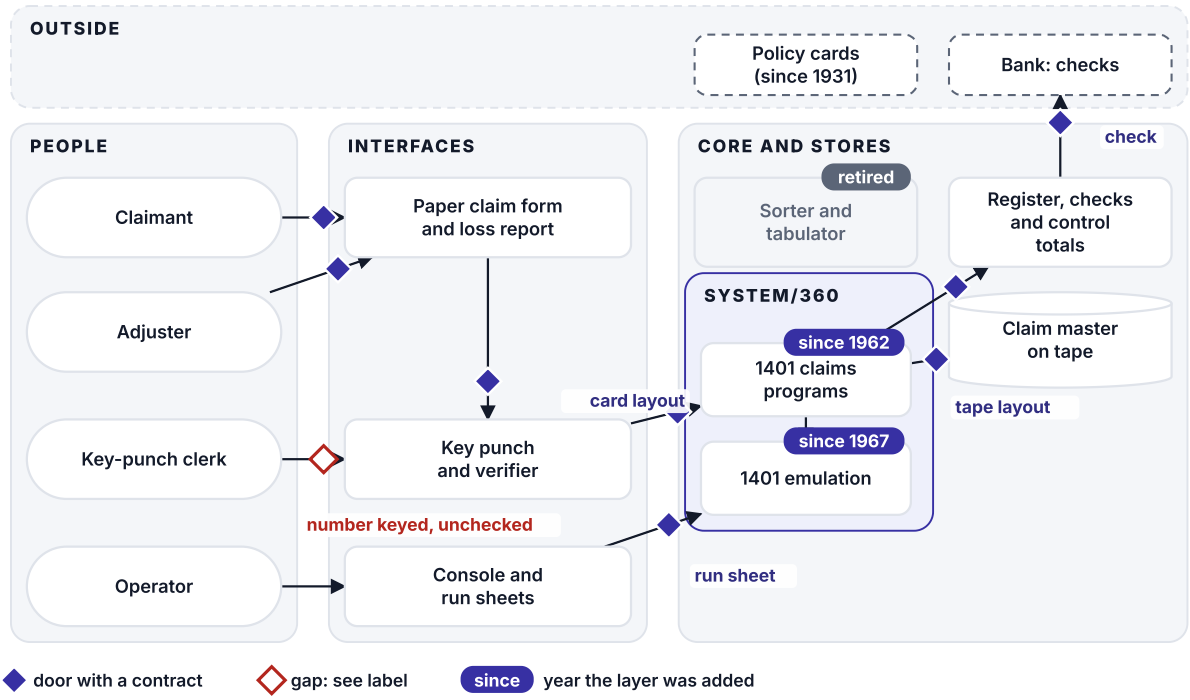
The insurer converted to System/360 in 1967, three years after the announcement. It did not rewrite its claims programs first. They kept running under emulation, unchanged, while new work was written for the new machine. This was the insurer's first wrap: an old component kept alive behind a new boundary, so that the business could change the platform without changing the logic at the same time.

THE CLAIMS SYSTEM · 1967 · SYSTEM/360 CONVERSION, 1401 PROGRAMS UNDER EMULATION

What was added	The claims run moves to an IBM System/360; the 1401 programs keep running unchanged under emulation, the insurer's first wrap.
Door and its contract	The emulated 1401 machine: old programs run as they were, behind the new architecture
In 2026	Retired; replaced by the 1974 core

FIGURE 4.1

1962 to 1967: a stored-program run, then the first wrap



In 1962 the insurer moved the claims run to a 1401-class batch; in 1967 it converted to System/360 and kept the 1401 programs running unchanged under emulation, its first wrap.

Author's diagram of the invented insurer, based on IBM's 1401 history and IBM System/360 Model 30 1401 Compatibility Feature, A24-3255-1 (April 1964).

Figure 4.1 shows both steps on the fixed canvas. The card tabulator has retired. The 1962 claims programs sit inside a System/360 box, running on the emulation feature. The card layout still marks the input; the tape layout marks the claim master; the keyed number is still unchecked. Under the wrap, the old programs still depended on the 1401's instructions; that dependence now sat in the emulator, where it stayed until the insurer replaced the programs in 1974 (Chapter 5). Martin Fowler named the piece-by-piece replacement of an old system around its edges the "Strangler Application" in June 2004, and later renamed it the strangler fig application.

The crisis: when programs outgrew their builders

System/360 needed an operating system, OS/360, and building it proved hard. IBM names Fred Brooks as the System/360 project leader, and an IBM history describes the "enormously difficult problems for Fred Brooks and his programming team". Brooks published what he learned in 1975, and Chapter 5 takes it up. The lesson of the decade was already visible: a program could be too big for the methods used to build it.

From 7 to 11 October 1968, a conference sponsored by NATO met at Garmisch, Germany, under the title *Software Engineering*. Its report, edited by Peter Naur and Brian Randell, appeared in January 1969. It records "a considerable amount of debate on what some members chose to call the 'software crisis' or the 'software gap'". The participants described late projects, unreliable systems and costs no one could estimate. A second conference followed in Rome from 27 to 31 October 1969; its report, edited by J. N. Buxton and B. Randell, appeared in April 1970.

At Garmisch, Doug McIlroy proposed one way out. "My thesis," he wrote, "is that the software industry is weakly founded, and that one aspect of this weakness is the absence of a software components subindustry." He wanted catalogs of routines that could be chosen by size, precision and speed, as engineers chose standard parts. "I would like to see components become a dignified branch of software engineering." The subroutine library of EDSAC (Chapter 3) had become a proposal for an industry. Its descendants run through this book: the shared library, the object, the service and the package registry.

Structure: the shape of the program and of the team

Two short texts of 1968 gave the crisis its structural vocabulary. In March 1968, *Communications of the ACM* printed a letter from Edsger Dijkstra under the title "Go To Statement Considered Harmful". Dijkstra's own typescript was titled "A Case against the GO TO Statement". His argument was that a program's text should show its progress: a reader should be able to tell, from where they are in the text, where the computation is. Unrestricted jumps destroyed that link. Structured programming followed, and with it the idea that a program's shape could be checked by reading it.

In April 1968, *Datamation* published Melvin Conway's "How Do Committees Invent?". Its thesis, in Conway's words: "organizations which design systems (in the broad sense used here) are constrained to produce designs which are copies of the communication structures of these organizations." The boundaries in a system follow the boundaries between the teams that build it. The insurer's systems already showed it: the claims programs belonged to the data processing department, the forms to the claims office, and the card layout sat on the line between them. Conway's paper returns in Chapter 9, when teams and services are drawn together on purpose.

The data itself was also getting structure. Charles Bachman's Integrated Data Store at General Electric, begun in 1961 by his own account, let programs navigate records linked to one another instead of reading files in order. IBM's IMS printed its first "READY" message at the Rockwell Space Division in Downey, California, on 14 August 1968, IBM announced the system that year, and IBM's documentation says it became available, renamed IMS/360, in 1969. IBM first shipped its Customer Information Control System in April 1968, as Public

Utility CICS, "free code, known as a Type II program" in IBM's own account; the CICS program product followed on 8 July 1969. The insurer adopted IMS for its open claims in 1974 and CICS in 1982. Bachman's own story, and how data models grew, is told in *From Tally to Token*.

WHAT HASN'T CHANGED

Platform changes since 1967 have offered several paths: rewrite the old logic, recompile or port it, buy a package, or run it unchanged behind a layer that imitates the old platform. The insurer took the last path in 1967. Emulators, virtual machines, containers holding old runtimes and adapters in front of old transactions are later forms of it, and in each the old platform's rules live on inside the layer. Chapter 12 lists the insurer's wraps still in place in 2026.

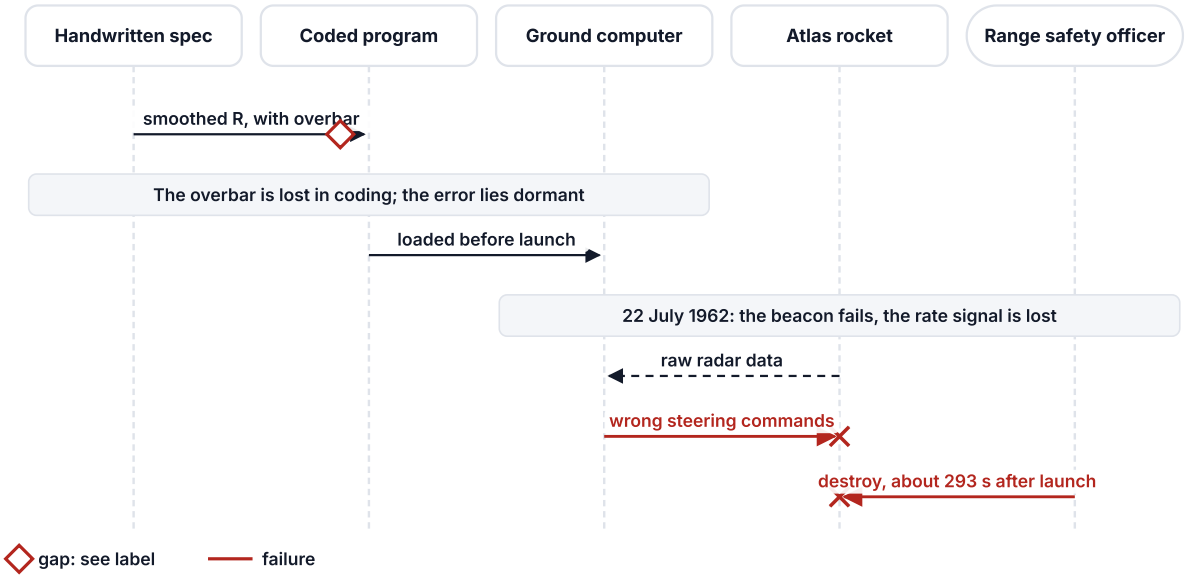
THE ERA'S SILENT FAILURE

Mariner 1, 22 July 1962. NASA's space science data archive gives two causes. "Improper operation of the Atlas airborne beacon equipment resulted in a loss of the rate signal from the vehicle for a prolonged period." Without that signal the ground guidance computer had to work from raw radar data. In that mode, "the omission of a superscript overbar when translating the handwritten equations into the coded computer instructions allowed transmission of incorrect guidance signals". NASA's history office identifies the missing mark as "an overbar for the symbol R for radius"; without it, NSSDCA says, the computer sent "instantaneous values instead of smoothed values". NSSDCA records that the Range Safety Officer destroyed the vehicle 293 seconds after launch.

The flight's end was loud. The silent part was the program: the coding error gave no sign while the beacon worked, because the faulty path ran only after the beacon failed. NASA's published records say where the mark was lost, in translating the handwritten equations into code; they do not describe how the code was checked against the equations. The error crossed from specification to program without raising anything.

FIGURE 4.2

Mariner 1: a coding error that stayed silent until a beacon failed



NASA records that an overbar was lost when the handwritten guidance equations were turned into code; the error stayed hidden until a beacon failure made the ground computer use raw radar data.

Author's diagram based on NASA, NSSDCA Mariner 1 record, and NASA History, 60 Years Ago: Mariner 1 Launch Attempt to Venus.

SECURITY LENS

Who can change it. In a batch shop, changes came through program decks kept by the programming group, run by operators who did not edit them; the tape library and the compatibility initialization deck that switched a Model 30 into 1401 mode sat with operations. Time-sharing and online systems such as SABRE added a new kind of person who could change records: anyone at a terminal with the right access. **How fast it is fixed.** A batch error was fixed before the next run; an online error had to be caught while the customer waited, or it spread to every agent reading the same record. **What AI changes.** An emulator reproduced the old machine's whole instruction set and restricted nothing: it was a compatibility layer, not a permission boundary. A wrap that lets an agent reach an old core is asked to restrict, and its list of permitted operations is the record of what it lets through.

DECISION CARD · DC-1967-01

1967 **Retired** Owner: Data processing manager Supersedes: DC-1962-01

1 Decision	Convert to the supplier's new compatible machine family and run the old claims programs unchanged under emulation until they are rewritten.
2 Outcome and binding constraint	Claims keep paying through the conversion, and no program is rewritten under deadline. **Binding constraint:** Programmer time: rewriting every program for the new instruction set would take years.
3 Boundaries and doors	**Old programs to new machine**: the emulated old instruction set **New programs to the files**: the same tape and card layouts **Operators to the run**: a run sheet for each emulated job
4 Options rejected	**Rewrite every program first**: too slow, and the old machine's rental cannot run on **Stay on the old machine**: no growth path once its supplier moves on
5 Quality · Security · Visibility · Cost	**Quality.** Emulated runs must give the same totals as the old machine. **Security.** Unchanged: the same decks, now behind the emulation. **Visibility.** Parallel runs compared total by total. **Cost.** Emulated programs use the new machine less efficiently; the rewrite is only deferred.
6 How it fails, and how we would see it	Nobody schedules the rewrite; the emulated programs stay on while the people who know them retire. **Evidence:** A count of programs still emulated, reviewed each year.
7 Run	Operators set the machine to 1401 compatibility mode with an initialization deck for each emulated run.
8 Reversal cost	High to go back once the old machine is gone; low to go forward, one rewritten program at a time.
9 Assumptions and Revisit when	The emulation stays supported for as long as the old programs need it. **Revisit when:** When the claims system is redesigned, or support for the emulation ends.
10 Change guide	1. Do not change an emulated program; replace it. 2. A rewritten program must reproduce the old totals before the old one retires.

Illustrative card for the invented insurer: its first wrap. Retired in 1974, when the core (DC-1974-01) replaced the emulated programs.

Monday rule: for one wrapped system you run, write down the exact operations the wrap lets through and the date by which the thing inside it must be replaced or re-justified.

CHAPTER FIVE

The 1970s: Structure and Change



On 9 November 1979, screens at command posts of the North American Air Defense Command showed what looked like a Soviet missile attack. There was no attack. The General Accounting Office later found that "test scenario data was inadvertently fed into the online missile warning computers which generated false alerts". This chapter is about building systems that can change without breaking, and about what a system trusts in the data that reaches it. The insurer's 1974 decision was to share one record layout among every program in its new core.

By 1970 the pressure lay elsewhere. Machines were cheaper and compatible; the large programs written for them in the 1960s were now running the business. The scarce thing was the ability to change those programs safely. Every new rule, product or regulation meant changing code that someone else had written, often years before. The decade's landmark papers are about structure because structure decides what a change will touch.

The insurer builds its core, 1974

In 1974 the insurer replaced its emulated 1401 programs with a new claims system written in COBOL for a System/370, under IBM's virtual-storage operating system, OS/VS, on which VSAM ran. The machine, the operating system, the language and the database all changed in the same year, so the new core ran beside the emulated programs until its totals matched theirs. It ran every night as a batch: job control language (JCL) set the steps, sort steps put the day's claims in order, and programs read and wrote sequential and VSAM files. The open-claim masters went into IMS/VS, the virtual-storage version of IMS. The nightly cycle created the claim of record, the insurer's official version of each claim, and recalculated the booked reserves, the money set aside for what open claims will cost. In 2026 it still does both.

The products predate the core. COBOL's specification dates from 1960 (Chapter 3), and IMS was announced in 1968 and available in 1969 (Chapter 4). IBM had published a general information manual for IMS/VS by January 1973, and its first planning guide for VSAM, the

Virtual Storage Access Method, is dated July 1972, the same summer IBM announced virtual memory across System/370.

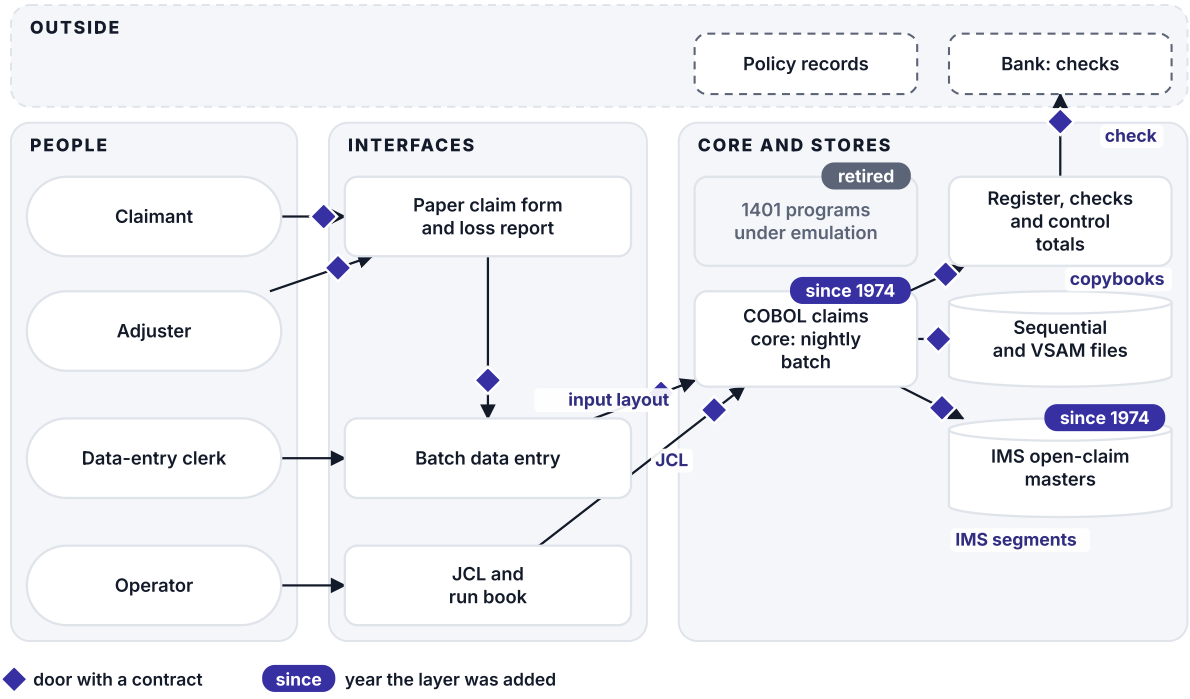
The insurer's reason for IMS was the shape of open claims. They are the records that change most: payments, reserve changes and notes arrive against each claim for months or years. A hierarchy fits them, a claim at the top and its payments and reserve entries beneath it. The insurer kept those active records in the database and left the rest of the claim history in files.

THE CLAIMS SYSTEM · 1974 · COBOL CLAIMS CORE, NIGHTLY BATCH

What was added	A COBOL batch system (JCL, sort steps), written for a System/370 under OS/VS, runs every night over sequential and VSAM files (the 1974 batch file); the open-claim masters are kept in IMS beside them. The claim of record is created in the nightly cycle, and booked reserves are recalculated there.
Door and its contract	COBOL copybooks (the record layouts every program shares) and the IMS segment layouts
In 2026	Still runs every night: it still creates the claim of record and recalculates booked reserves

FIGURE 5.1

1974: the COBOL core. Every program shares the copybooks, so every layout is a contract for all of them



The insurer's 1974 claims core on the fixed canvas: a nightly COBOL batch over sequential and VSAM files, open-claim masters in IMS, and record layouts shared through copybooks; the 1401 programs and their emulation retire.

Author's diagram of the invented insurer, based on IBM OS/VS VSAM Planning Guide, GC26-3799-0 (July 1972), and IMS/VS General Information Manual, GH20-1260-0 (January 1973).

The boundary that matters most in Figure 5.1 is the copybook. A copybook is a piece of COBOL source that describes a record layout: which bytes hold the claim number, the date, the amount. Every program that reads a claim record copies the same copybook into its own source. The layout is defined once, and every program agrees on it. That is the strength of the design and its cost. Change a copybook in a way that moves or resizes a field, and every program that reads that record must be recompiled; the insurer's rule was to recompile and test all of them together. The card layout of 1956 had become a shared text, and it bound every program that read it.

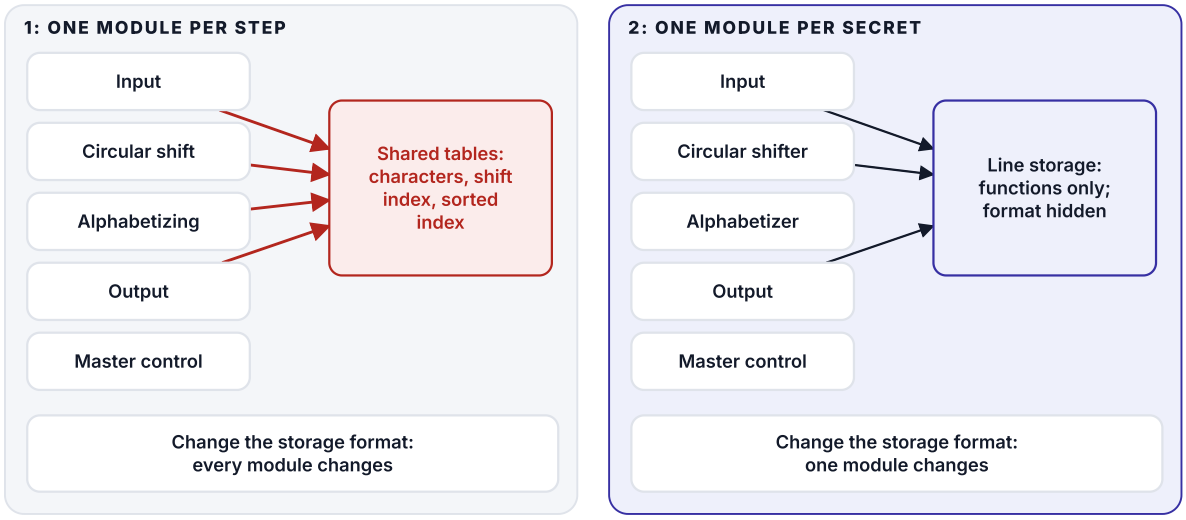
Modules and the secrets they keep

In December 1972, *Communications of the ACM* published David Parnas's "On the Criteria To Be Used in Decomposing Systems into Modules". Parnas took a small program, a KWIC index that lists every circular shift of a set of lines in alphabetical order, and divided it two

ways. The first followed the steps of processing: input, shift, sort, output. The second followed the decisions most likely to change. His criterion was plain: "one begins with a list of difficult design decisions or design decisions which are likely to change. Each module is then designed to hide such a decision from the others."

FIGURE 5.2

Parnas's two decompositions: by processing step, or by the decision each module hides



In the first modularization every module reads one shared storage format, so changing that format touches them all; in the second, the format is a secret of one module behind its functions.

Author's diagram based on D. L. Parnas, On the Criteria To Be Used in Decomposing Systems into Modules, Communications of the ACM 15(12), December 1972.

Figure 5.2 shows the difference. In the first decomposition every module knows how lines are stored, so a change to the storage format touches all of them. In the second, one module owns the format and the others ask it for what they need. Applied to the insurer, the copybook is Parnas's first decomposition: every program knows the record layout. It was a reasonable choice in 1974: calling through a module for every field cost machine time the nightly window did not have. The cost arrived later, from 1997 to 1999, when Y2K remediation had to find every two-digit year in every copybook and every program that used it.

Two papers of 1974 carried the idea forward. Barbara Liskov and Stephen Zilles's "Programming with Abstract Data Types" (*SIGPLAN Notices*, April 1974) made a hidden representation part of the language: a type defined by its operations, not its layout. Wayne

Stevens, Glenford Myers and Larry Constantine's "Structured Design" (*IBM Systems Journal*, 1974) gave designers words for how modules depend on each other. "Coupling is the measure of the strength of association established by a connection from one module to another," they wrote.

Method, people and the cost of being late

In August 1970, Winston Royce presented "Managing the Development of Large Software Systems" at IEEE WESCON. The paper drew a sequence of steps from requirements to operations, the diagram later called the waterfall. Royce did not recommend it as drawn. "I believe in this concept, but the implementation described above is risky and invites failure," he wrote, and the rest of the paper adds feedback, a preliminary design and a step headed "Do it twice": a first pass that "provides an early simulation of the final product".

In 1975 Fred Brooks published *The Mythical Man-Month* (Addison-Wesley), drawing on his years leading System/360 and OS/360. Its best-known line is his own summary: "Oversimplifying outrageously, we state Brooks's Law: Adding manpower to a late software project makes it later." New people must be trained, and every pair of people who must talk adds a channel. Across the industry the scarcer thing was now the time and attention of the people who understood the code; machine time still mattered, but it no longer decided alone.

Meir Lehman measured that constraint. With L. A. Belady he studied how OS/360 changed from release to release; their 1976 paper in the *IBM Systems Journal* already stated "laws of program evolution". In 1980 Lehman restated the laws in the *Proceedings of the IEEE*. The first: "A program that is used and that as an implementation of its specification reflects some other reality, undergoes continual change or becomes progressively less useful." The second: "As an evolving program is continually changed, its complexity, reflecting deteriorating structure, increases unless work is done to maintain or reduce it." The insurer's core has changed every year since 1974.

Data, transactions and machines that do not stop

In June 1970, Edgar Codd of IBM published "A Relational Model of Data for Large Shared Data Banks" in *Communications of the ACM*. Programs, he argued, should ask for data by its values, not navigate the paths a database used to store it. Applications would then survive changes to the way data was stored. IBM's System R project tested the idea; its own history dates "Phase Zero" to "1974 and most of 1975". The relational model reached the insurer only in 1987, as DB2 beside the core (Chapter 6).

Online systems raised a different question: what happens when a machine fails in the middle of a change? Tandem Computers, established in November 1974, delivered its first NonStop system in May 1976, according to a Datapro industry report of 1979. It was designed, as its name said, to keep running when a part of it failed. Jim Gray's "Notes on Data Base Operating Systems", published by Springer in 1978, described how locks and recovery logs make a transaction all-or-nothing. The insurer's nightly batch had a simpler answer: if a step failed, restart it from its checkpoint before morning. That answer worked as long as nobody needed the data during the night.

Why the core lasted

The System/360 paper (Chapter 4) had made a promise with conditions: a valid program "whose logic will not depend implicitly upon time of execution and which runs upon configuration A, will also run on configuration B if the latter includes at least the required storage, at least the required I/O devices, and at least the required optional features". When IBM announced System/370 on 30 June 1970, it promised customers they could move up "without having to reprogram the vast majority of their existing System/360 applications". Through later IBM hardware generations, the insurer did not rewrite its core to change machines. The platform still brought work: new operating system releases, compiler upgrades and database versions each meant recompiling and retesting programs, though not redesigning them.

The other part is the cost of replacement. In Lehman's 1980 table of laws, the first law ends with a sentence that the quotation above leaves out: the change or decay process "continues until it is judged more cost effective to replace the system with a recreated version". For the insurer, that judgment has not yet come out in favor of replacing the core. It tried once, from 2006 to 2009, and abandoned the rewrite (Chapter 8). Chapter 11 treats the other side of this ledger, the rigidity those shared copybooks imposed.

> **OUR VIEW, AS OF NOVEMBER 2026**
>
> The core is often said to have survived because nobody dared touch its copybooks. That is part of the answer. We think the larger part is IBM's compatibility boundary, which spared the core the kind of instruction-set change that had pushed the 1401 programs into emulation, together with the cost of replacing a run that balances every night.

THE ERA'S SILENT FAILURE

NORAD, 9 November 1979. The GAO found that test scenario data had been "inadvertently fed into the online missile warning computers", which processed it as warnings. The alert itself was anything but quiet. What made no sound was the crossing: the GAO's account records no check that stopped test data at the door into the live system. The GAO noted that "since testing is no longer performed on the live 427M system, this type of false alert should not recur".

JUAN'S LENS

Parnas's criterion is the one I use most: list what is likely to change, then put each of those things behind a boundary that one owner controls. The insurer's copybooks did the opposite, and in 1974 that was the right call for a machine with no time to spare. What I take from the decade is not that the copybook was wrong. It is that the shared layout should have been written down as a decision, with its owner and its cost of change, so that the Y2K team found a record, not an archaeology site.

SECURITY LENS

Who can change it. A Defense Science Board task force chaired by Willis Ware set out *Security Controls for Computer Systems* in a report published by RAND in February 1970. In March 1973 trading in Equity Funding Corporation of America's securities was halted. A 1974 opinion of the Judicial Panel on Multidistrict Litigation recites the allegations: "creating and selling to reinsurers bogus life insurance policies", a fraud "facilitated by the use of computers". Saltzer and Schroeder's "The Protection of Information in Computer Systems" (*Proceedings of the IEEE*, September 1975) gave the field principles that bear on such a case, among them "least privilege" and "separation of privilege": a mechanism that needs two keys, or two conditions, before it grants access. At the insurer, operations owned the job control, the programming group owned the programs and the copybooks, and a change went in after the programming manager signed it. **How fast it is fixed.** In a nightly batch, a bad record is fixed in the next run, if anyone sees it; a fraud designed to balance is not seen. **What AI changes.** An agent that can both write records and write the report about them would hold, at machine speed, two powers that a separation of duties keeps in different hands; a two-key rule in Saltzer and Schroeder's sense is one way to enforce that split.

DECISION CARD · DC-1974-01

1974 **Accepted** Owner: Head of claims systems Supersedes: DC-1967-01

1 Decision	Create the claim of record and recalculate booked reserves in one nightly batch over shared record layouts, with the open-claim masters in a hierarchical database.
2 Outcome and binding constraint	Every claim registered by the next morning, reserves booked once, and one set of totals the auditors can check. **Binding constraint:** The time of the few people who understand the old programs, and machine time in one nightly window; online access for every handler costs too much.
3 Boundaries and doors	**Programs to files**: the copybooks: one record layout shared by every program **Programs to open claims**: the database segment layouts **Run to finance**: the control totals and the payment record
4 Options rejected	**Keep the emulated programs**: each change needs skills the shop is losing **Online entry for every handler now**: terminal and machine cost beyond the budget **Keep rewriting one by one**: few were rewritten since 1967; each kept old file designs
5 Quality · Security · Visibility · Cost	**Quality.** Before cutover, the new run's totals must match the emulated programs' totals in parallel; after it, totals balance every night or the run stops. **Security.** Only operations changes the job control; a program changes only after the programming manager signs the change. **Visibility.** The control report printed after every run. **Cost.** Machine hours per night; a copybook change means recompiling every program that uses it.
6 How it fails, and how we would see it	A copybook changes and a program that was not recompiled reads a character field from the wrong positions without stopping. **Evidence:** Totals that do not balance; the claims office's reconciliation next morning.
7 Run	Operations runs it nightly; a failed step restarts from its checkpoint.
8 Reversal cost	High: every program that shares the copybooks must change together.
9 Assumptions and Revisit when	One nightly window is long enough; next-day registration is acceptable. **Revisit when:** When claimants or regulators expect same-day answers, or the window no longer fits.
10 Change guide	1. Never change a copybook without recompiling every program that uses it. 2. The control totals must balance before any payment leaves the run.

Illustrative card for the invented insurer. Still in force in 2026: put online in 1982, reached by a three-tier desk in 1997; a rewrite started in 2006 was abandoned in 2009.

Monday rule: list the shared record layouts in your oldest system, name an owner for each, and record what a change to it would force you to recompile, retest and redeploy.

CHAPTER SIX

The 1980s: Computing Leaves the Glass Room



On 12 August 1981, at the Waldorf Hotel in New York, IBM unveiled the IBM Personal Computer. IBM's own history gives the price as USD 1,565, for a machine with 16 kilobytes of memory and no disk drive. It was small, and it sat on a desk instead of behind the glass wall of the computer room. Within a few years, the people who had waited months for the data processing department could do some of the work themselves. This chapter follows that escape. It also follows the decision the insurer made in 1982 to put claims online without moving the claim of record out of the nightly batch.

The queue outside the glass room

By the early 1980s, most large companies ran their core work on one mainframe in one room, with glass walls, raised floors and an operations team. The machine was still the most expensive thing in the building, and its time was still rationed, a second constraint behind the binding ones of the 1960s and 1970s, the complexity of large projects and the ability to change a program. The designs of those years answered them well: one nightly batch, shared record layouts, and control totals that the auditors could check.

The constraint that ran out first in the 1980s was different. It was the queue of requests waiting for the data processing department. A claims manager who wanted a new report, a new screen or a change to a calculation joined a list, and the list grew faster than the department could clear it. James Martin's *Application Development Without Programmers* (Prentice-Hall, 1982) named it the application backlog in its first chapter: "There is an application backlog of years." The machine was no longer the scarcest thing. The scarce thing was the small group of people allowed to change what ran on it.

Two answers came out of that pressure, and the rest of the chapter follows both. The first was to bring the users closer to the central machine, through terminals and online transactions. The second was to give users a machine of their own, on the desk. The first kept control in the glass room. The second let work leave it.

The insurer took the first path in 1982. Since 1974 its claims had been registered and its reserves recalculated in a nightly COBOL batch over sequential and VSAM files, with the open-claim masters in IMS (Chapter 5). In 1982 it put claim inquiry and entry online through CICS on 3270 terminals. CICS was not new. IBM first shipped it in April 1968 as PUCICS, a free program for public utilities; IBM then dropped "Public Utility" from the name and released the CICS product on 8 July 1969, "designed to support 50 BTAM terminals" in its first form. The 3270 display system was on the market by May 1971 at the latest, the date of the second edition of IBM's introduction to it. The insurer adopted CICS thirteen years after the product's release; its year is an adoption year, not a release year.

THE CLAIMS SYSTEM · 1982 · CICS ONLINE INQUIRY AND ENTRY

What was added	Claims staff inquire and enter claims on 3270 terminals through CICS transactions over the core files.
Door and its contract	CICS transaction codes and the screen maps
In 2026	Still runs

The design of the 1982 decision is the worked card at the end of this chapter. Its central rule is one line: an entry made on a screen waits for the night, and only the nightly batch creates the claim of record. A handler could now tell a caller where a claim stood, and enter a new claim while the caller waited. The claim did not exist officially until the batch had run that night, checked its totals and written it. The batch's totals still balanced, because there was still exactly one writer of the claim of record.

Measured against the backlog and the shared machine, the choice held. Online settlement would have meant rewriting the reserving logic and the control totals for many concurrent writers, on a machine whose daytime capacity was shared with every other department. Staging online and settling at night bought most of the benefit for a fraction of the risk. It also created a seam that still runs through the insurer in 2026: what a handler sees on the screen at three in the afternoon is not yet the record. The morning reject list, which shows the entries that the batch refused, is where that seam becomes visible.

A computer on the desk

The second answer arrived without asking permission. The Computer History Museum describes VisiCalc, "a 1979 spreadsheet program by Software Arts", as the "killer app" that spurred Apple II sales. The IBM PC followed in 1981. Lotus 1-2-3 was being shown to user

groups by early 1983: the museum holds a recording of Lotus presenting it to the Boston Computer Society on 23 February 1983. A manager with a PC and a spreadsheet no longer needed to join the backlog to see a total, a trend or a what-if.

For a claims office, this changed the shape of the system without changing a line of the core. Figures moved from the mainframe to the desk on paper. An analyst received a printed report, typed the figures into a spreadsheet, and built a model that the core never saw. The spreadsheet became a new boundary in the claims system, and it had no agreement on its edge: no shared layout, no control total, no reconciliation back to the source. If a figure was keyed wrongly, nothing would notice. The card shop of the 1950s had at least balanced its control totals. The 1980s desk had nothing equivalent.

It would be easy to read this as a mistake by the people who did it. By the constraint of the decade, it was the opposite. The backlog was real, and the spreadsheet cleared some of it in an afternoon. The insurer, for its part, had no sanctioned way to move numbers from the core to the desk without retyping them. That gap is drawn as the hollow red marker between the reports and the analyst in Figure 6.1.

Relational data beside the core

Another change of the decade happened inside the glass room. Edgar Codd's relational model had been published in the Communications of the ACM in June 1970 (Chapter 5). IBM's history records that Codd received the ACM Turing Award in 1981, and that IBM Db2 "was announced in 1983, and V1.1 launched on MVS in 1985". IBM marks 7 June 1983 as the day the product was born. The insurer adopted DB2 in 1987, four years after the announcement and two after the MVS release.

THE CLAIMS SYSTEM · 1987 · DB2 RELATIONAL DATABASE BESIDE THE CORE

What was added	A DB2 database beside the core holds reporting copies first, then a nightly copy of policy data; the core keeps its own files, and the DB2 policy tables are a copy, never the master.
Door and its contract	Reporting copies read through the core's copybooks; the nightly policy extract from policy administration's file, whose layout lives only in the extract program
In 2026	Still runs

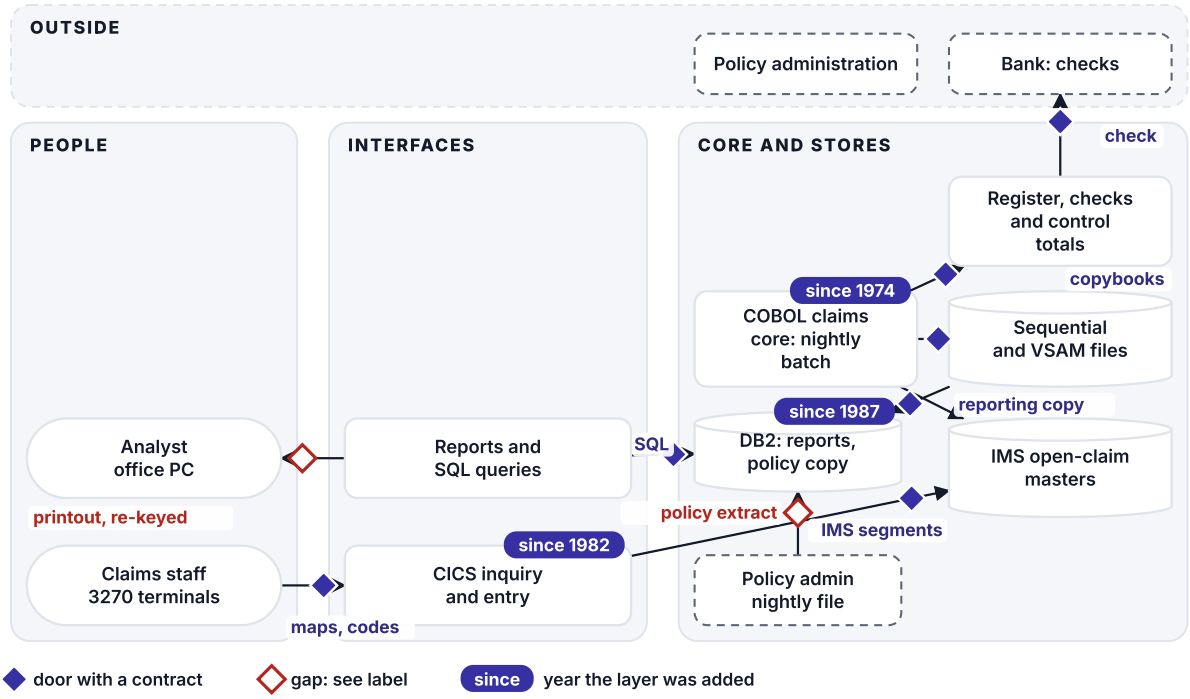
The insurer did not move its core into DB2. The core's COBOL programs, copybooks and control totals had been built and audited over thirteen years. DB2's value in 1987 was in answering questions that nobody had planned for, through SQL, without a programmer

writing a new batch report. So DB2 went beside the core, not under it. It held reporting copies of the core's own figures first, read from the core's files through the core's copybooks. Then it took a nightly copy of policy data from policy administration, a separate system run by another department and the master of policy data. The core kept its own files, and DB2's policy tables were a copy, never the master.

The policy extract is the second hollow red marker in Figure 6.1. Each night it reads the day's policy changes from a file that policy administration writes, and loads them into DB2 tables. Nothing written records what it promises: which fields, which day, what happens if a file is late. Its copy of the file's layout lives inside the extract program, written on the claims side, while the file belongs to another department with its own library and schedule. Its one check is a record count in the file's trailer. The written layouts and amount totals that the 1974 batch used on its own files were already in the building; the policy extract had a count but no total, and no layout that both teams had agreed. Chapter 1 shows what that cost in November 2025.

FIGURE 6.1

1987: a nightly policy extract with no written agreement fills DB2 beside the core



The insurer's claims system in 1987 on the fixed canvas: 3270 terminals reach the open-claim masters through CICS (adopted 1982; CICS product released 1969), DB2 (adopted 1987, announced 1983) takes reporting copies from the core's files and a nightly policy extract from policy administration's file, and figures reach office PCs only by re-keying.

Author's diagram of the invented insurer; layer years are the insurer's adoption years.

Networks, and the call that looks local

Once computers sat on many desks, they needed to talk to each other. Two pieces came together in 1983. On the wide-area side, Jon Postel's RFC 801 (November 1981) set the plan for the ARPANET: "The goal is to make a complete switch over from the NCP to IP/TCP by 1 January 1983." On the local side, Robert Metcalfe and David Boggs had described Ethernet in the Communications of the ACM in July 1976, and the IEEE standards board approved the 802.3 standard on 23 June 1983. A building could now have a local network, and the local network could reach other networks.

Networks raised a design question that the industry is still answering. How should one program ask another program on a different machine to do something? The most influential answer of the decade came from Andrew Birrell and Bruce Nelson at Xerox, in "Implementing Remote Procedure Calls" (ACM Transactions on Computer Systems,

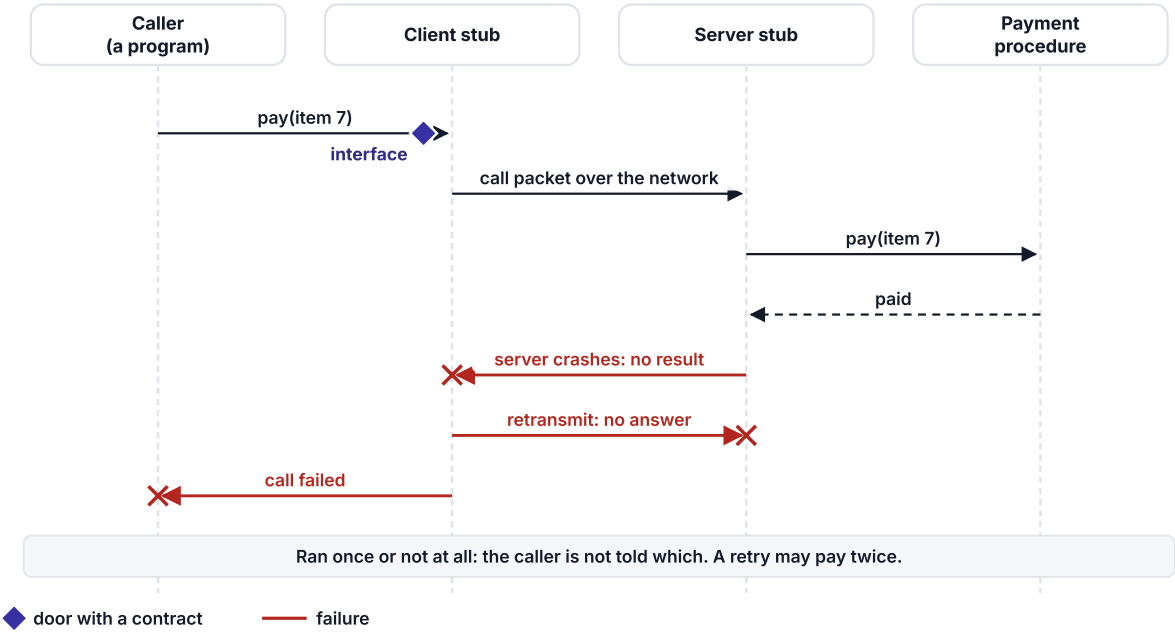
February 1984). A remote procedure call, or RPC, lets a program call a procedure on another machine with the same syntax as a local call. A generated piece of code on each side, the stub, packs the arguments into a message and unpacks them at the other end.

Birrell and Nelson stated the aim plainly: "the semantics of remote procedure calls should be as close as possible to those of local (single-machine) procedure calls." They were equally plain about where the resemblance ends. "We guarantee that if the call returns to the user then the procedure in the server has been invoked precisely once. Otherwise, an exception is reported to the user and the procedure will have been invoked either once or not at all", and the user "is not told which." They add that a "call failed" exception "is the primary way in which our clients note the difference between local and remote calls."

The "call failed" case is where the resemblance to a local call ends. Within one machine, a call and its caller usually fail together. A remote call can fail halfway: the server may have done the work and crashed before it answered. Figure 6.2 draws the case for a payment. The caller sees "call failed", and the paper's own rule says the payment ran once or not at all. If the caller simply tries again, the payment may be made twice. The figure is the paper's general case, not a system the insurer ran in the 1980s. The insurer's branch PCs would reach CICS through an RPC gateway program from 1991, and its claims desk would call CICS through the same gateway from 1997 (Chapter 7).

FIGURE 6.2

A remote call looks like a local call until the server fails: then the caller cannot know whether the work ran



Birrell and Nelson's rule (1984), drawn for a payment of item 7: a call that returns ran precisely once; a call that fails ran once or not at all, the caller is not told which, and a retry may pay twice.

Author's diagram based on Birrell and Nelson, "Implementing Remote Procedure Calls", ACM TOCS 2(1), 1984.

In November 1984, Jerome Saltzer, David Reed and David Clark published "End-to-End Arguments in System Design" in the same journal. Their argument is about where to put a function such as reliable delivery or duplicate suppression: "The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the end points of the communication system." For the payment in Figure 6.2, that means the network cannot decide whether a retry is safe. Only the payment program, which knows whether item 7 has been paid already, can. The end-to-end paper was a revised version of one first presented in April 1981, at the Second International Conference on Distributed Computing Systems in Paris. Read together, the two papers of 1984 cover both halves: one made remote calls easy, and the other explained why the ends must still check.

Why computers stop, and what cannot be bought

Three essays, two from the second half of the decade and one delivered just after it, give its engineering mood. The first is Jim Gray's "Why Do Computers Stop and What Can Be Done About It?", a Tandem technical report dated June 1985 (TR 85.7, revised November 1985). Gray analyzed the failures of Tandem's fault-tolerant systems reported to Tandem over a seven-month period, from more than 2,000 systems and over 1,300 system years: 166 failures in all. His abstract states that "administration and software are the major contributors to failure". The percentages come after he set aside the "infant mortality" failures of new products, about a third of the total, leaving 107 (p. 5). Of those, system administration, which includes operator actions, configuration and maintenance, "was the main source of failures", at 42 percent, and software faults were "a major source of system outages", 25 percent in all. Gray was careful about the sample: "The statistics for fault-tolerant systems are quite a bit different from those for conventional mainframes", because these machines were built to mask hardware faults, and he suspected that operator failures were under-reported (pp. 5 to 6).

Gray's remedies are still quoted. His rule was to "make each module fail-fast": either it does the right thing or it stops. Pair each important process with a backup, so that when one stops cleanly, the other takes over. Wrap work in transactions, so that a stopped process leaves no half-done change behind. All three, in Gray's account, make a failure stop and show instead of carrying on.

The second essay is Frederick Brooks's "No Silver Bullet", given at the IFIP congress in 1986 and reprinted in IEEE Computer in April 1987. Brooks separated the essential difficulty of software, the complexity of the thing being specified, from the accidental difficulty of the tools. His central claim is worth quoting whole: "There is no single development, in either technology or management technique, which by itself promises even one order-of-magnitude improvement within a decade in productivity, in reliability, in simplicity." For a decade that hoped fourth-generation languages and personal computers would clear the backlog, it was an unwelcome forecast.

The third is Richard Gabriel's "worse is better" argument. Gabriel's own account says it began as a keynote he was invited to give in the summer of 1989, was delivered at Cambridge University in March 1990, and was republished in AI Expert in June 1991. Gabriel contrasted a design that is correct and complete with one that is simple to implement and ships early. He observed that the simple one tended to spread, and then improve. In his words, "Early Unix and C are examples of the use of this school of design."

Put side by side, the three essays describe the decade's trade. Simple things spread faster than correct ones (Gabriel). No tool would make the hard part go away (Brooks). And in fault-tolerant systems, at least, the failures that remained came mostly from people and

software, not machines (Gray). Gray's own remedies, modules that stop cleanly and work wrapped in transactions, were the decade's engineering answer to that last point.

THE ERA'S SILENT FAILURE

Therac-25, 1985 to 1987. The Therac-25 was a computer-controlled radiation therapy machine. Nancy Leveson and Clark Turner's investigation in IEEE Computer (July 1993) begins: "Between June 1985 and January 1987, six known accidents involved massive overdoses by the Therac-25", with deaths and serious injuries among the results.

The machine's builders had decided "not to duplicate all the existing hardware safety mechanisms and interlocks" of its predecessor; "software checks were substituted for many traditional hardware interlocks" (pp. 20, 25). "Race conditions resulting from this implementation of multitasking played an important part in the accidents" (p. 21): an operator who edited the treatment data quickly could produce the error, because "data-entry speed during editing was the key factor" (p. 28).

The failure reached the operator as a message that said almost nothing. In one accident the console "displayed the message 'Malfunction 54.'" It was later found to be a "dose input 2" error, meaning a dose "either too high or too low" had been delivered, and the dose monitor display showed "a substantial underdose": "6 monitor units delivered, whereas the operator had requested 202 monitor units" (p. 27). According to an FDA memorandum, "the operator's manual supplied with the machine does not explain nor even address the malfunction codes" (p. 24). The pause could be resumed, and the operator "immediately took the normal action when the machine merely paused, which was to hit the 'p' key to proceed" (p. 27).

The silent part is exact. The machine had delivered an overdose and its own display reported an underdose; nothing told the operator that the treatment had gone wrong in the other direction. A later accident at Yakima had a different software cause: a one-byte counter meant that "on every 256th pass through Set-Up Test, the upper collimator will not be checked" (p. 34). There, the check on the upper collimator was skipped and raised no error of its own; what the operator saw was a pause, which he cleared with the "P" key, and then "flatness" on the reason line (p. 33). Among the contributing factors, the authors list "overconfidence in the software and removal of hardware interlocks" (p. 38).

JUAN'S LENS

The seams in this chapter fail in three different ways. Therac-25 was silent where it mattered: its display reported an underdose after an overdose, and its error message was a code that the operator's manual did not explain. The spreadsheet said nothing at all. The remote call said exactly what it knew, "call failed", which is honest and still not enough to act on. When I look at an inherited system, I sort its seams into those three kinds before I change anything, because each needs a different repair: a message a person can act on, a reconciliation, or a way to ask the far side what really happened.

SECURITY LENS

Who can change it. In the glass room, the answer was short: the operations team and the programmers who passed the change board. Online transactions added a second answer, the users who could sign on and run a given transaction code. The PC added a third: anyone with a spreadsheet could change a model that drove a business decision, with no record of who changed what. Networks added a fourth: a program from outside. In November 1988 a self-copying program spread across the Internet and "reportedly infected up to 6,000 computers" according to the GAO's report of June 1989, a press figure; the GAO also cited a researcher's estimate of 1,000 to 3,000. **How fast it is fixed.** A fault in the core waited for the next release and the next batch window. A wrong formula on a desk was fixed as fast as someone noticed it, which, without a reconciliation, could be never. **What AI changes.** An agent that drafts a spreadsheet, a report or a query is the 1983 desk again, faster: useful work outside the central change process. The 1980s remedy carries over: the agent's output can be reconciled to the system of record as a careful analyst's would be, and the reconciliation helps only if a person or a check reads it.

DECISION CARD · DC-1982-01

1982 **Accepted** Owner: Head of claims systems Supersedes: none (adds to DC-1974-01)

1 Decision	Put claim inquiry and entry online through a transaction monitor over the core's records, and keep the claim of record in the nightly batch.
2 Outcome and binding constraint	Handlers answer a caller's question and take a new claim while the caller waits; the night's totals still balance. **Binding constraint:** A request backlog the data processing department cannot clear, and a shared machine whose nightly window must stay untouched.
3 Boundaries and doors	**Terminal to transaction**: transaction codes and screen maps **Transaction to core records**: the same record layouts the batch uses; online may read and stage, never settle **Staged entries to nightly batch**: an entries file with control totals, and a reject list the next morning
4 Options rejected	**Settle claims online**: reserves and totals assume one nightly writer of the claim of record **A separate minicomputer for the claims office**: a second copy of claim data with no reconciliation to the core
5 Quality · Security · Visibility · Cost	**Quality.** Response time per transaction is measured; the batch totals still balance every night. **Security.** Each user signs on; the sign-on table decides which transaction codes a user may run; only the batch creates the claim of record. **Visibility.** Daily transaction counts; the batch's accept and reject list compared with the day's entries. **Cost.** Terminals, lines and a daytime share of the machine; one more system to run.
6 How it fails, and how we would see it	An entry accepted on screen is rejected by the batch at night and nobody keys it again. **Evidence:** The morning reject list matched against the day's entry log, entry by entry.
7 Run	Operations owns online availability, the batch cutoff, and reconciliation of staged entries with the next eligible batch.
8 Reversal cost	Medium: the screens can be withdrawn, but staff work and volumes soon depend on them.
9 Assumptions and Revisit when	Same-day answers are enough; same-day creation of the claim of record is not needed. **Revisit when:** Callers or brokers need the claim of record created while they wait, or the batch no longer fits its window.
10 Change guide	1. Screens may read and stage; only the nightly batch creates the claim of record. 2. Every staged entry appears on its eligible batch's accept or reject list.

Illustrative card for the invented insurer. Still in force in 2026.

Monday rule: List every place where people or programs copy numbers from one system into another, and give each one a reconciliation against its source that someone is named to read.

CHAPTER SEVEN

The 1990s: Integration and the Web



On 30 April 1993, CERN put the software of the World Wide Web, available by FTP since 1991, into the public domain. Its declaration read: "CERN relinquishes all intellectual property rights to this code, both source and binary form." Four years later the insurer had a claims desk on three tiers, a payment queue and, beginning that same year, a program of work to find every two-digit year in its core. None of it was the web yet. All of it was integration: making systems that had grown apart work as one. This chapter follows the decisions of that decade of connection.

From two tiers to three

By 1990, an insurer like this one had two kinds of computing that did not meet. In the glass room ran the core, the CICS transactions and DB2. On the desks, and now in the branches, ran PCs on local networks. The decade's binding constraint was the cost of joining the two: each new link between a desk program and the mainframe was custom work, and the mainframe side changed slowly because so much depended on it.

The first common answer was client-server: a program on the PC (the client) talks over the network to a program that owns the data (the server). The insurer's version arrived in 1991. Branch PCs reached the existing CICS transactions through an RPC gateway program that the insurer wrote itself, in the remote procedure call style that Birrell and Nelson had described in 1984 (Chapter 6). The gateway did not change CICS. On the mainframe side it drove each transaction as a 3270 terminal would, so a remote call from a PC became the same transaction a terminal user would have run.

THE CLAIMS SYSTEM · 1991 · RPC GATEWAY FOR BRANCH PCS

What was added	Branch PCs reach the CICS transactions through an RPC gateway program.
Door and its contract	The gateway's remote procedure interface
In 2026	Still runs

That made the interface screen-shaped. A PC program that wanted a claim's status had to ask for it the way a terminal user would: one transaction code, one screen's worth of fields, in the order of the screen map. The interface was stable because the screens were stable, and it was clumsy for the same reason. Every new use of claim data had to fit into a transaction that was designed for a person at a terminal.

The next step was a middle tier. In a three-tier design, the client handles presentation, an application server holds the business logic, and the database or the legacy system holds the data. The insurer built its three-tier claims desk in 1997: a Windows client for handlers, an application server, and DB2. The application server exposed claim objects, each with methods such as "status" or "add note", and behind those methods it still called CICS transactions through the 1991 gateway. The desk wrapped the core. It did not replace any part of it.

THE CLAIMS SYSTEM · 1997 · THREE-TIER CLAIMS DESK

What was added	A Windows client, an application server and DB2 give claims handlers a desk application; the application server exposes claim objects over CORBA IIOP and calls CICS transactions through the 1991 gateway program.
Door and its contract	The CORBA interfaces (IDL) of the claim objects and the calls into CICS
In 2026	Still runs; it reaches CICS through the 1991 gateway, not the bus

The wrap fitted the constraint, and the worked card at the end of this chapter records why. A rewrite of the core was out of reach: the core team was about to spend three years on the year 2000 problem. Letting the new tier read the core files directly would have bypassed the validation in the CICS transactions and the shared layouts. A wrap gave handlers one desk with claim, policy copy and payments in one place, while leaving the claim of record exactly where the 1974 batch put it.

Objects across the wire

The middle tier needed a way for the client to call the claim objects. The standard the insurer chose was CORBA, the Common Object Request Broker Architecture of the Object Management Group (OMG). OMG's history dates CORBA 1.0 to October 1991, and its specification catalog lists August 1991 for the same version. CORBA 1.0 already included the Interface Definition Language, IDL: a neutral description of an object's methods and their parameters, from which tools generate code for each language. CORBA 2.0 added the Internet Inter-ORB Protocol, IIOP, so that brokers from different vendors could talk to each other. The 2.0 specification is dated July 1995, updated July 1996, and OMG's history page lists it under August 1996. The insurer adopted CORBA IIOP in 1997.

IDL was a real agreement on a real boundary, and tools checked it. A client calling through the stubs generated from the claim object's IDL could not call a method that did not exist, or pass text where the IDL declared a number. IDL checked declared types, not meaning: an amount and a date declared with the same numeric type could still be swapped. What IDL could not settle at all was failure. In November 1994 Jim Waldo, Geoff Wyant, Ann Wollrath and Sam Kendall at Sun Microsystems Laboratories set out why that mattered, in "A Note on Distributed Computing" (technical report SMLI TR-94-29).

Their argument was that a remote object is not a local object with a longer wire. "We argue that objects that interact in a distributed system need to be dealt with in ways that are intrinsically different from objects that interact in a single address space." A distributed program, they wrote, must "be aware of latency, have a different model of memory access, and take into account issues of concurrency and partial failure." And they were blunt about systems that pretend otherwise: "work in distributed object-oriented systems that is based on a model that ignores or denies these differences is doomed to failure".

A typed interface describes what a call looks like. CORBA did give the caller a signal for the half-done case of Figure 6.2: every standard exception carries a completion status, and COMPLETED_MAYBE means "The status of implementation completion is indeterminate" (CORBA 2.0, section 3.15). What to do next, retry or first ask the core whether the work ran, was left to each application, as Birrell and Nelson had left it in 1984. CORBA itself later declined. Michi Henning wrote in 2006 that it had become "a niche technology", and named the OMG's "technology adoption process" as "the core reason for CORBA's decline", with the complexity of its APIs as the most obvious technical problem (ACM Queue, June 2006).

Another short list from Sun, the fallacies of distributed computing, made the same point. Its best early copy, a page on James Gosling's site at Sun archived in 1999, credits it to Peter Deutsch: "Essentially everyone, when they first build a distributed application, makes the following eight assumptions", beginning with "The network is reliable" and "Latency is zero". In a 2021 interview, Deutsch said that the first four came from one of the Lyon

brothers at Sun and were already known when he arrived there, where he worked from late 1991 to early 1993, and that he added the next four. Some later accounts credit parts of the list to Bill Joy or James Gosling instead.

Two books of the decade gave designers a shared vocabulary for what they were building. Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides published Design Patterns: Elements of Reusable Object-Oriented Software, which Addison-Wesley lists as published on 31 October 1994 with a 1995 copyright. Philippe Kruchten's "4+1" view model, in IEEE Software in November 1995, proposed describing an architecture through several views (logical, process, development, physical) tied together by scenarios. Both were ways of writing down a design so that someone else could reason about it later.

Messages instead of calls

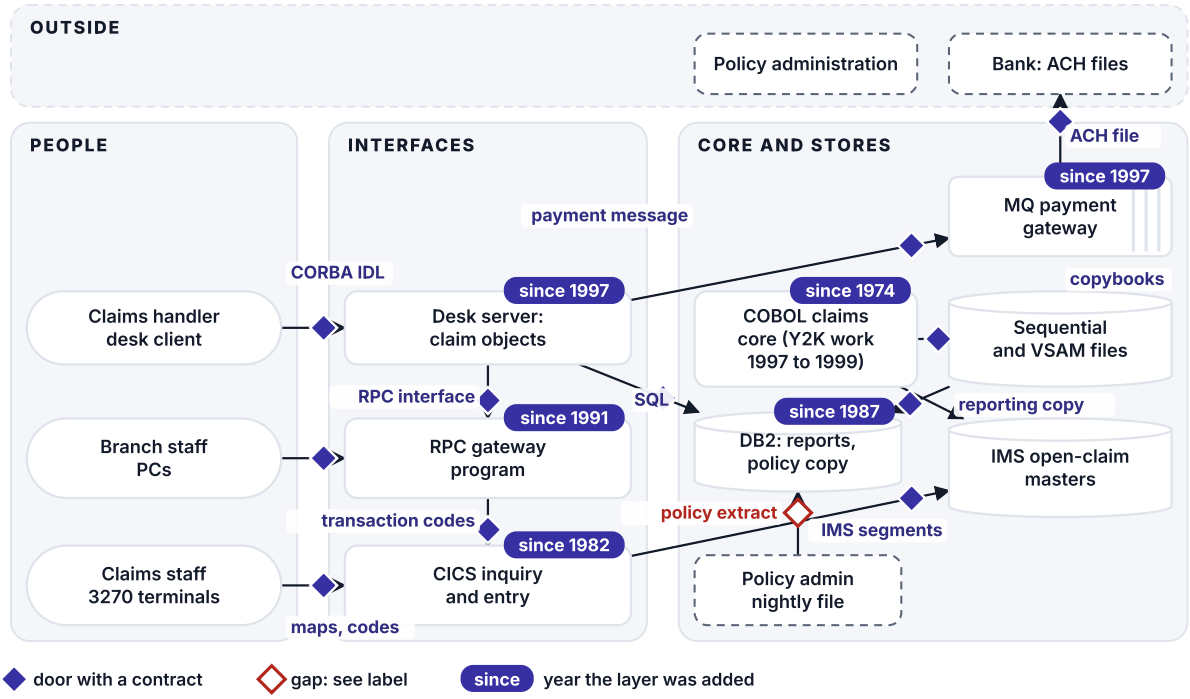
A remote call waits for an answer. A message does not. In message queuing, one program puts a message on a queue, and another program takes it off when it is ready. If the receiver is down, the message waits. IBM's MQSeries brought it to mainframe shops. IBM describes its messaging product as "Originally introduced in 1993 (under the IBM MQSeries name)".

The insurer used a queue for the decision where waiting mattered least and losing a message mattered most: money. Until 1997, claims were paid by check. In 1997 payments moved to electronic funds transfer. The claims desk put each payment request on an MQ queue, and a payment gateway took the requests off it. For each one, the gateway recorded the payment against the claim with one CICS transaction, as a staged entry that the nightly batch posted to the claim of record, so the 1982 rule held. It then sent ACH files to the bank. The bank's file format and the payment request message were the two agreements on that path. Taking a message and recording the payment had to succeed or fail together, or a payment could be sent without a record, or recorded twice. The era's general tool for that was a unit of work spanning the queue and the database, coordinated by two-phase commit. The ACH file to the bank sat outside any such unit, so the payments recorded in the core still had to be reconciled with what the bank received.

The queue changed what a failure looked like. If the gateway or the bank link was down, payment requests waited on the queue instead of failing on a handler's screen. That is a gain, and it creates a new question: who notices when the queue is growing and nothing is coming off it? Messaging as a discipline is covered in depth later in this series.

FIGURE 7.1

1997: the three-tier desk wraps the core in claim objects, but still reaches CICS through the 1991 gateway



The insurer's claims system in 1997 on the fixed canvas: branch PCs (1991) and the new three-tier desk (1997) both reach CICS through the RPC gateway program, and the desk queues payment requests to an MQ gateway that sends ACH files.

Author's diagram of the invented insurer; layer years are the insurer's adoption years.

The web, and a common syntax

The web did not reach the insurer's claims system in the 1990s, but it set the terms for the next decade. Tim Berners-Lee's proposal at CERN is dated March 1989, with the date May 1990 added when it was redistributed. Berners-Lee announced on the alt.hypertext newsgroup on 6 August 1991 that the code was "available by anonymous FTP from info.cern.ch". The same post called the code "copyright CERN but free distribution and use is not normally a problem", the position that the 1993 release changed. The Common Gateway Interface, the simple convention by which a web server runs a program and returns its output, has, in the words of its later specification (RFC 3875, October 2004), been "in use by the World-Wide Web (WWW) since 1993", as an interface "developed and documented at the U.S. National Center for Supercomputing Applications". HTTP/1.0 was documented as RFC 1945 in May 1996, and HTTP/1.1 as RFC 2068 in January 1997.

Sun announced the Java language and the HotJava browser on 23 May 1995, the opening day of SunWorld, and released the "Java 1.0 programming environment" on 23 January 1996. On 10 February 1998, XML 1.0 became a W3C Recommendation. XML gave everyone the same syntax for structured data: tags, nesting and a way to declare which tags are allowed. It did not give them the same meaning. Two companies could exchange perfectly valid XML and still disagree about what an amount or a date meant.

A written agreement on a boundary, and no check that enforces it, is the subject of this chapter's silent failure.

THE ERA'S SILENT FAILURE

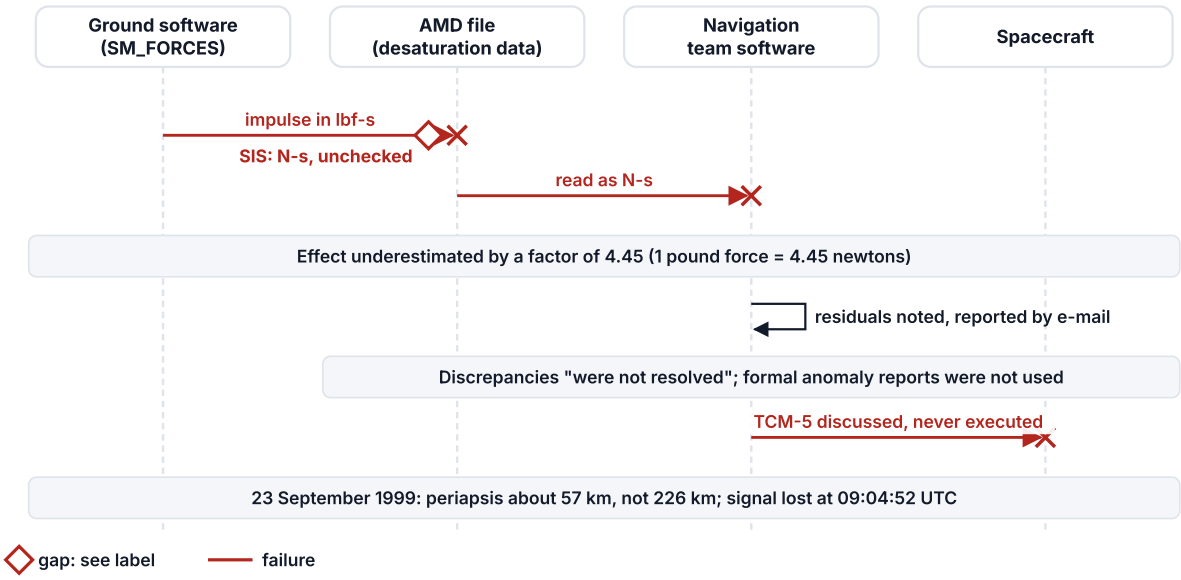
Mars Climate Orbiter, 23 September 1999. NASA lost the orbiter as it arrived at Mars. Its Mishap Investigation Board reported on 10 November 1999 and named the root cause in one sentence: "the failure to use metric units in the coding of a ground software file, 'Small Forces,' used in trajectory models" (Executive Summary, p. 6). The full story of the loss is told in *From Tally to Token*; this chapter keeps only the boundary that failed.

The boundary ran between two organizations: Lockheed Martin Astronautics in Denver performed spacecraft operations, and JPL was responsible for navigation (p. 10). It had a written agreement. "The output from the SM_FORCES application code as required by a MSOP Project Software Interface Specification (SIS) was to be in metric units of Newton-seconds (N-s). Instead, the data was reported in English units of pound-seconds (lbf-s)." The board adds: "The SIS, which was not followed, defines both the format and units of the AMD file generated by ground-based computers" (p. 16); AMD stands for angular momentum desaturation, the small thruster firings the file recorded. The navigation software therefore "underestimated the effect on the spacecraft trajectory by a factor of 4.45, which is the required conversion factor from force in pounds to Newtons" (p. 16).

The silent part was the file. It was produced, accepted and used for months, and nothing at the boundary checked its units against the specification. People did see symptoms: within about a week of the files becoming usable in April 1999, "it became apparent that the files contained anomalous data" (p. 18), but the team "relied on e-mail to solve problems, instead of formal problem resolution processes" (p. 22). Among eight contributing causes, the board lists that the "verification and validation process did not adequately address ground software" (p. 7). The written specification was right. Nothing at the file checked that the file obeyed it.

FIGURE 7.2

Mars Climate Orbiter: the interface specification said newton-seconds, and no check at the file enforced it



The "Small Forces" ground software wrote impulse to the file in pound-seconds where the Software Interface Specification required newton-seconds; no check rejected the file, trajectory models underestimated the effect by a factor of 4.45, and the doubts that surfaced were never resolved.

Source: Mars Climate Orbiter Mishap Investigation Board, Phase I Report, NASA, 10 November 1999, pp. 6 to 7, 13 to 25.

Two digits

The decade's largest project at the insurer was not integration at all. It was inside the core. The US General Accounting Office explained the problem in its September 1997 assessment guide (GAO/AIMD-10.1.14): "For the past several decades, systems have typically used two digits to represent the year, such as '97' representing 1997, in order to conserve on electronic data storage and reduce operating costs. With this two-digit format, however, the Year 2000 is indistinguishable from 1900, 2001 from 1901, and so on."

The two-digit year was not carelessness. In 1974 storage was expensive, and a two-digit year in every date of every record saved disk and tape across files that grew for twenty-five years. The design was right for its binding constraint. What changed was the calendar.

Remediation used a small set of techniques. IBM's planning guide (GC28-1251, ninth edition, February 1998) lists conversion to a full four-digit year, compressed date data, windowing, and a two-digit encoding scheme. Its windowing chapter says: "Two types of windowing techniques have been defined: the fixed window technique and the sliding

(rolling) window technique." A fixed window "uses a static 100-year interval that generally crosses a century boundary", so that, for example, years 00 to 49 are read as 2000 to 2049. IBM called this solution "temporary", because applications might later process dates more than 100 years apart. The GAO guide adds the practical warning that converting large databases "from 2-digit to 4-digit year fields is a time consuming effort", and recommends "bridges or filters to convert non-conforming data" at interfaces.

The insurer ran its Y2K remediation of the core from 1997 to 1999. Its business let it start that late: property and casualty policies mostly run for one year, so few dates beyond 1999 reached the claims core before 1999 itself. Its programmers found the two-digit year fields in the copybooks and files, and fixed them by expansion where a file could be converted and by windowing where it could not. The shared copybooks made the work both possible and painful. Possible, because a date field was defined once and every program that used it could be found. Painful, because every program that used it had to be recompiled and retested together. The rigidity of shared copybooks is examined as a case where a principle cost too much in Chapter 11.

JUAN'S LENS

The two boundaries in this chapter fail in opposite ways. The orbiter's file had a written specification and nothing that checked traffic against it. The claims desk had IDL, checked by the compiler and the broker, and still left "maybe it ran" to whoever wrote the client. When I inherit an interface, I ask two separate questions: what rejects a message that breaks the agreement, and what does the caller do when the answer is "maybe"? A check at the queue that rejects malformed messages answers the first for structure; for meaning it takes more, such as the unit carried with each value and a test that the producer really converts it. A payment reference the far side can look up answers the second.

SECURITY LENS

Who can change it. Three tiers spread the answer across three teams: the desk client, the application server and the mainframe. The application server called CICS under its own service identity, so the core saw one caller where it had once seen each handler. Whoever could change the server could act as every handler. **How fast it is fixed.** The middle tier could be patched in days; the core still moved at the pace of its batch releases, and the Y2K years froze most other core change. A fix that needed both ends moved at the speed of the slower one. **What AI changes.** An agent that calls tools is a new middle tier that speaks for many users. The 1997 desk shows the question it raises: whether the core sees the agent or the person behind it, and whether the trail of each call records both.

DECISION CARD · DC-1997-01

1997 **Superseded** Owner: Claims systems architect Supersedes: none (wraps DC-1982-01)

1 Decision	Wrap the core behind claim objects on an application server for a three-tier desk, and reach the online transactions through the 1991 remote-call gateway.
2 Outcome and binding constraint	Handlers see claim, policy copy and payments on one desk without any change to the core. **Binding constraint:** Integration: screen-shaped interfaces everywhere, and a core team busy with year-2000 work until 1999.
3 Boundaries and doors	**Desk client to application server**: the claim objects' interface definitions **Application server to online transactions**: the gateway's remote call interface and the transaction codes **Desk to payment gateway**: the payment request message on the queue
4 Options rejected	**Rewrite the core in the new tier**: the year-2000 fixes already fill the core team **Let the server read the core files directly**: it would bypass the online validation and the shared layouts
5 Quality · Security · Visibility · Cost	**Quality.** Desk response time per remote call is measured; only the core creates claims; every payment sent has a record in the core. **Security.** Users sign on to the server; the server calls the core under one service identity, so the core no longer sees the person. **Visibility.** Server logs per remote call; the core sees one caller. **Cost.** A middle tier to run and staff, and its licenses.
6 How it fails, and how we would see it	A remote call times out after the transaction ran; the desk shows an error and the handler enters it again. **Evidence:** A failed remote call is checked against the core before anyone enters it again; each morning, the server log's failed calls are matched against the core's records.
7 Run	An application server team; the gateway restarts with the online region.
8 Reversal cost	Medium: the desk can be withdrawn, but claims work now runs through it.
9 Assumptions and Revisit when	The screen-shaped transactions stay stable; one gateway carries the load. **Revisit when:** A second channel, web or partner, needs the same claim functions.
10 Change guide	1. Claim objects reach the core only through the gateway, never its files. 2. A failed remote call is checked against the core before anyone retries it.

Superseded as the target design by DC-2007-01. Superseded means no longer the target, not switched off: the desk still runs in 2026.

Monday rule: For each interface you depend on, find the written specification and the code that checks traffic against it; where only the first exists, add the second.

CHAPTER EIGHT

The 2000s: Services and the Team



On 30 July 2002, the US Sarbanes-Oxley Act became law (Public Law 107-204). Its Section 404 directed the Securities and Exchange Commission to require that each annual report contain "an internal control report", including "an assessment" of "the effectiveness of the internal control structure and procedures" for financial reporting. The duty fell on companies that file annual reports with the SEC, and the Act did not mention software. It still reached every system of those companies that moved money or booked a reserve, because those systems were now part of the controls that management had to stand behind. Insurance regulators followed: in 2006 the NAIC amended its model audit regulation, a model that binds only where a state adopts it, so that an insurer with $500 million or more in annual premiums, listed or not, would report on its internal control over financial reporting, for periods ending from 31 December 2010. An insurer of that size could see internal control reporting coming. For the insurer, the decade brought a web portal, a service bus, an outsourced mainframe and a rewrite of the core that was started and then stopped. This chapter follows the bus that survived and the rewrite that did not.

The portal, and the services behind it

The decade's binding constraint was the speed of change across many connected systems. Each new channel, partner or rule touched several systems built in different decades, and each link between them was custom. The 1990s had connected things. The 2000s had to change connected things without breaking them.

The insurer opened a web portal for brokers and customers in 2003. Brokers and customers could now see and file claims without calling a handler. The portal's first door into the system was the desk's application server: it called the same claim objects the handlers used.

THE CLAIMS SYSTEM · 2003 · WEB PORTAL FOR BROKERS AND CUSTOMERS

What was added	Brokers and customers see and file claims on the web.
Door and its contract	The portal's calls to the desk's application server
In 2026	Still runs

The industry's answer to linking many systems was the web service, and then the service-oriented architecture, or SOA. A web service exposed a business function over the web's protocols, described in a machine-readable form. Two W3C Notes set the form. SOAP 1.1 (W3C Note, 8 May 2000) called itself "a lightweight protocol for exchange of information in a decentralized, distributed environment". WSDL 1.1 (W3C Note, 15 March 2001) was "an XML format for describing network services as a set of endpoints operating on messages". With WSDL, a service's operations and message formats were written down in a file that a tool could read and check.

Insurers had an industry body for the meaning that XML alone did not carry (Chapter 7). ACORD, today the Association for Cooperative Operations Research and Development, was announced in September 1970 with an initial focus on property and casualty insurance, and its own history places its first property and casualty XML standards in the 1990s. For a carrier and its brokers, ACORD's messages became the shared payload on the portal and the bus: the same claim, policy and party structures on both sides.

SOA's central piece in many companies was the enterprise service bus, or ESB: a middleware layer through which services and channels call each other, with routing, transformation of message formats and central logging. The term came into wide use in the early 2000s; its exact origin is disputed. David Chappell's Enterprise Service Bus (O'Reilly, June 2004) gave it a book-length definition. The insurer built its ESB in 2007, as part of an SOA program, several years after the term spread (the year is the insurer's adoption year).

THE CLAIMS SYSTEM · 2007 · ENTERPRISE SERVICE BUS

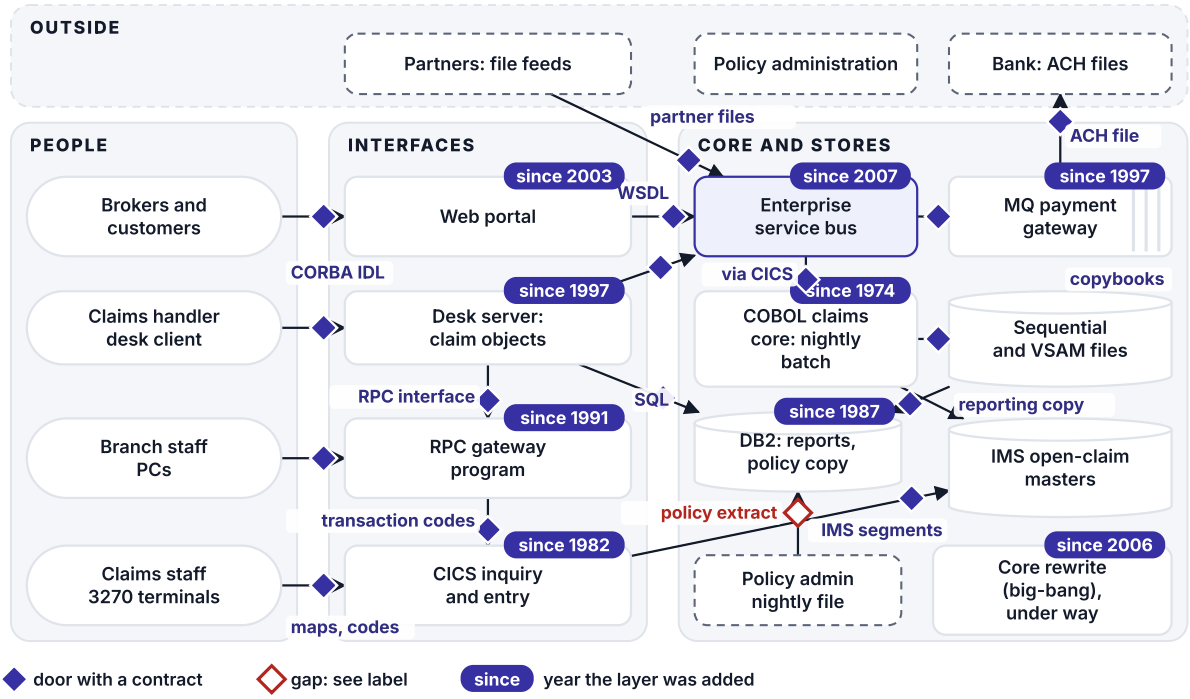
What was added	An ESB connects the portal, the desk, the core and partners under an SOA program; it survived the abandoned 2006 to 2009 rewrite.
Door and its contract	Service contracts (WSDL) and the bus's routing rules
In 2026	Partly retired. It still carries portal inquiries to the core, payment instructions to the bank and the general ledger interface, older partner file feeds, and the policy-coverage lookup used by an orphaned deployment of the estimate service's 2019-06 version, which still runs (covered in depth later in this series). Broker traffic (the 2013 broker API, beside the bus), intake, the initial reserve estimate and the reinsurer feed have moved off it.

The bus connected the portal, the desk, the core and the partners. Portal inquiries reached the core through bus services that called CICS transactions. Partner file feeds from loss adjusters and repair networks came onto the bus, and so did the feed to the insurer's reinsurer. Payment requests, which since 1997 had gone from the desk onto the MQ payment queue, now traveled on the bus as payment instructions. The desk itself kept calling CICS through the 1991 gateway: the bus wrapped the core a second time, beside the first wrap. The desk kept running, but new channels were built on the bus, not on the desk's server, which is why the 2007 card supersedes the 1997 one as the target design.

The ESB decision is the worked card at the end of this chapter. Judged by its decade's constraint, the bus answered it. Each new channel or partner now needed one link to the bus instead of links to every system it touched. Every service had a published description. Every message from the portal and the partners passed one place on its way to the core where it could be logged, and a single path for payment instructions could be reconciled with the bank's files each day. That last point mattered as internal control reporting came in for insurers. The bus also repeated a weakness of 1997: like the desk, it called the core under one service identity. Standards for carrying the end user's identity inside a message were available by then, among them OASIS's Web Services Security 1.0 (March 2004); the bus did not pass that identity on to the core. The card names the other risk: one central team owned every change on the bus, and when most changes needed the bus, that team's queue became the speed limit. How the bus later became the bottleneck, and what replaced parts of it, is covered in depth later in this series.

FIGURE 8.1

2007: the portal and the bus in front of a core that a rewrite did not replace



The insurer's claims system in 2007 on the fixed canvas: the 2003 portal and the partners reach the core through the 2007 service bus, which also carries payment instructions; the desk still reaches CICS through the 1991 gateway; the core rewrite is shown under way, as it stood in 2007 (it was stopped in 2009).

Author's diagram of the invented insurer; layer years are the insurer's adoption years.

Two other facts of the decade sit around the figure. From 2001, medical bills for motor and liability claims arrived through a clearinghouse in the X12 837 format, the health care claim standard that the US Department of Health and Human Services adopted in its final rule of 17 August 2000, with compliance due from 16 October 2002 for health plans and providers (small health plans by 16 October 2003); the Administrative Simplification Compliance Act of 27 December 2001 let a covered entity move that date to 16 October 2003 by filing a compliance plan. The rule did not bind the insurer: a health plan under HIPAA excludes liability insurance and automobile medical payment insurance (45 CFR 160.103), so the insurer took the format by agreement with its clearinghouse, not by law. And from 2004, the insurer's mainframe operations were run by an outsourcer, not by its own staff: the glass room still existed, but the people in it worked for someone else.

The web's own architecture

While enterprises built buses, a different description of distributed systems was already in print. Roy Fielding's doctoral dissertation at the University of California, Irvine (2000), "Architectural Styles and the Design of Network-based Software Architectures", defined the style he called REST. In his summary of how the style is derived (section 5.1.8): "REST consists of a set of architectural constraints chosen for the properties they induce on candidate architectures." The constraints are client-server, stateless interaction, cacheable responses, a uniform interface, a layered system, and optional code on demand.

REST and SOAP answered different questions. SOAP and WSDL let an enterprise describe a remote operation precisely and call it like a procedure, the same move Birrell and Nelson had made in 1984. REST described how the web itself had scaled: many independent parties, a few uniform operations on resources, and caches in between. Inside one company, with one bus team and a governed catalog, the insurer used SOAP services on its bus. The decade used both, and the insurer's later broker API (2013) belongs to the REST family.

The rewrite that stopped

In 2006 the insurer started a rewrite of its claims core. The idea was to replace the 1974 COBOL batch, its copybooks and its nightly creation of the claim of record with a new system that would create claims online. The decision record is DC-2006-01 below: the outcome it served, the constraint it answered, and the ways it might fail.

The constraint was real. Every change to the core waited for programmers who could read shared copybooks, for a test cycle across every program that used them, and for the nightly window. The coming internal control rules added a demand for documented controls over the flows that fed financial reports. A new core promised to answer both at once.

DECISION CARD · DC-2006-01

2006 **Reversed** Owner: Chief information officer Supersedes: none (meant to replace DC-1974-01)

1 Decision	Replace the claims core in one big-bang program with a new system that creates the claim of record online.
2 Outcome and binding constraint	Same-day claims, and one place to change a rule instead of layouts shared by every batch program. **Binding constraint:** Speed of change: each core change waits on shared layouts, scarce batch skills and the nightly window; coming internal control reporting for insurers asks for documented controls.
3 Boundaries and doors	**Old core to new core at cut-over**: a one-time conversion of every open claim and reserve **New core to portal and desk**: published service descriptions **New core to the auditors**: totals that match the old batch to the cent in every parallel run
4 Options rejected	**Wrap and replace piece by piece**: slower to show value; old and new parts run side by side for years **Buy a packaged claims system**: packaged claims systems were on the market, but tailoring their reserving rules looked close to a rewrite
5 Quality · Security · Visibility · Cost	**Quality.** No cut-over until parallel runs reconcile with the old batch. **Security.** A new access model, audited before go-live. **Visibility.** Milestones, and the breaks found in each parallel run. **Cost.** Two teams and two systems for years; the cost comes before any value.
6 How it fails, and how we would see it	Parallel runs keep breaking; the conversion of open claims fails a reconciliation; the old team cannot change two systems at once. **Evidence:** Reconciliation breaks per parallel run, and the count of rules found only in old programs.
7 Run	A program team builds; the old core team keeps the old system running and changes both.
8 Reversal cost	Technically low, since the old core never stopped; in money high: the spend is written off.
9 Assumptions and Revisit when	The old core's rules can be written down from documents and interviews; one cut-over weekend is enough. **Revisit when:** At every parallel run, and whenever the rule count still rises.
10 Change guide	1. The old core stays the claim of record until a parallel run balances. 2. Every rule found in old code is written down before it is rebuilt.

Reversed: the program was abandoned in 2009 and the old core kept running. The bus begun in 2007 survived (DC-2007-01).

The program was abandoned in 2009, and the 1974 core kept running. The bus, begun in 2007 to connect the portal, the desk, the core and the partners, did not depend on the rewrite and survived it. The card's status is therefore Reversed.

The insurer is invented; abandoned core replacements are not. The Co-operative Bank's, canceled in 2013 and examined in Sir Christopher Kelly's report of 30 April 2014, is told at length in a later book of this series.

Read by the constraint of 2006, the rewrite answered a real cost. The old core's cost of change was high and rising, and a big-bang replacement promised to reset it. The card listed several ways it could fail. In the insurer's story the first came true: each parallel run found rules that had been added to the old programs over thirty years and written down nowhere else, and the date moved with every run. The alternative the card rejected, wrapping and replacing piece by piece, already had a name: Martin Fowler had described it as the "Strangler Application" on 29 June 2004. It was slower to show value, which in 2006 counted against it. Chapter 12 returns to it.

Teams, domains and the circuit breaker

The second half of the chapter's title is the team. From 11 to 13 February 2001, seventeen people met at The Lodge at Snowbird in Utah and wrote the Manifesto for Agile Software Development. Its first value line reads: "Individuals and interactions over processes and tools." The manifesto moved attention from the plan to the team that builds and changes the software, in short cycles, with the customer close by.

Two books of 2003 gave teams a vocabulary for large systems. Eric Evans's Domain-Driven Design (Addison-Wesley; published 20 August 2003, copyright 2004) argued that the structure of software should follow the language of the business domain, and that a large system should be divided into bounded contexts, each with its own consistent model. Gregor Hohpe and Bobby Woolf's Enterprise Integration Patterns (Addison-Wesley; published 10 October 2003, copyright 2004) cataloged the patterns of messaging between systems: channels, routers, translators, and the rest of what a bus does. One book described where the lines should go; the other described what crosses them.

The third book named a failure mode that the bus made more likely. Michael Nygard's Release It! (Pragmatic Bookshelf, announced 30 March 2007) was about software in production, and its chapter of stability patterns includes the circuit breaker. Martin Fowler later wrote that in Release It!, "Michael Nygard popularized the Circuit Breaker pattern to prevent this kind of catastrophic cascade." A circuit breaker wraps a call to another system. When the calls keep failing, it stops making them for a time and fails at once, instead of letting every caller wait on a dead dependency until their own threads run out. In a system where the portal and the partners all reach the core through the same bus, one slow service can hold up everyone. The circuit breaker turns a slow, spreading failure into a fast, local and visible one.

At the end of the decade the team idea reached operations. At O'Reilly's Velocity conference on 23 June 2009, John Allspaw and Paul Hammond of Flickr gave the talk "10+ Deploys Per Day: Dev and Ops Cooperation at Flickr", and the first devopsdays conference met in Ghent on 30 and 31 October 2009. The next chapter follows where that led.

Someone else's computer

The decade's other change happened outside the insurer, and it would not reach the claims system until much later. Amazon opened its Simple Queue Service for beta testing on 3 November 2004 and moved it to production in July 2006. It launched the S3 storage service on 14 March 2006, available that day, and the EC2 compute service went into limited beta in late August 2006. Google launched a preview of App Engine on 7 April 2008. Microsoft announced Windows Azure on 27 October 2008 and declared it generally available on 1 February 2010. The vendors and their services are named here as history; the book's platform mapping, as of November 2026, is in Appendix D.

The new element was the shape of the cost and the contract. A company could rent a queue, storage or a machine by the hour through an API, with no purchase order and no room. The binding constraint of the 1960s, the cost of the machine, came back in a new form: not the price of the computer, but the bill for using it, which grows with every call.

Researchers and one large web company also published what they had learned about data at that scale. In a keynote at the PODC conference on 19 July 2000, Eric Brewer stated the conjecture that a shared-data system can have at most two of consistency, availability and tolerance to network partitions. Seth Gilbert and Nancy Lynch proved a formal version in ACM SIGACT News in June 2002. Brewer himself later warned against the slogan: "The '2 of 3' formulation was always misleading because it tended to oversimplify the tensions among properties" (IEEE Computer, February 2012). His later article argues that the choice arises only when a partition actually happens, and can be made differently for different operations.

Amazon's Dynamo paper (SOSP, October 2007) showed one such choice in production: a key-value store designed as "an 'always writeable' data store where no updates are rejected due to failures or concurrent writes", with conflicts resolved later. Amazon made that choice for its shopping cart. The insurer's core, which creates one claim of record and books reserves an auditor will check, stayed on its mainframe, now run by the outsourcer, with one writer and one night.

THE ERA'S SILENT FAILURE

The Northeast blackout, 14 August 2003. The US-Canada Power System Outage Task Force published its final report in April 2004. The outage "affected an area with an estimated 50 million people and 61,800 megawatts (MW) of electric load" (p. 1).

The failure that let a local problem grow was a monitor that stopped without telling anyone. "Shortly after 14:14 EDT, the alarm and logging system in FE's control room failed and was not restored until after the blackout" (p. 45). FE is FirstEnergy, the utility whose lines were failing that afternoon. "Neither FE's control room operators nor FE's IT EMS support personnel were aware of the alarm failure" (p. 52). The primary control system server hosting the alarm function failed at 14:41 EDT, and the backup at 14:54, "because the alarm application moved intact onto the backup while still stalled and ineffective" (pp. 52, 54).

The operators read the silence as good news. "They did not know that they were operating without alarms, so that they did not realize that system conditions were changing" (p. 52); they continued to work "without awareness that they were running the system without these alarms and should no longer assume that not getting alarms meant that system conditions were still safe" (p. 53). "For over an hour no one in FE's control room grasped that their computer systems were not operating properly" (p. 51). At 15:42 EDT, "the IT staff were first told of the alarm problem by a control room operator" (p. 56). Elsewhere, the regional reliability coordinator's state estimator had been "effectively out of service between 12:15 EDT and 16:04 EDT" because of inaccurate input data (pp. 45, 49).

The automatic pages went to IT, not to the control room. "FE's IT engineer was auto-paged" when the servers failed at 14:41 and 14:54 (p. 52), but "they did not notify control room operators of the problem" (p. 54). IT staff completed a "warm reboot" of the primary server at 15:08 and "did not confirm that the alarm system was again working properly with the control room operators" (p. 54).

The trip of the Sammis-Star 345-kV line at 16:05:57 EDT was "the event that triggered a cascade"; by 16:13, "more than 508 generating units at 265 power plants had been lost" (p. 74). The report groups the causes; one group is "Inadequate situational awareness at FirstEnergy" (p. 18). The servers had a page when they failed. The alarm function itself had none, and the pages went to people who did not tell the operators.

FIGURE 8.2

14 August 2003: the alarm system stopped, and the silence looked like a quiet grid



From 14:14 EDT FirstEnergy's alarm and logging software had failed; the server failures paged IT staff but not the operators, who began to recognize their situation only at about 15:42, and the cascade that cut power to about 50 million people followed from 16:05:57.

Source: US-Canada Power System Outage Task Force, Final Report on the August 14, 2003 Blackout, April 2004, pp. 1, 45 to 65, 74.

JUAN'S LENS

The bus in this chapter carried the same exposure: its central log was where trouble would show, and a log that stops writing looks like a quiet day. What strikes me in Figure 8.2 is that half of the monitoring worked: the servers failed and a pager went off. It went to the wrong people, and it watched the machine, not the function the operators relied on. When I review monitoring, I ask two questions: does the check watch the thing the user depends on, and does the signal reach the person who would act differently? I trust a dashboard only after I have seen it turn red on purpose, in front of the people who use it.

SECURITY LENS

Who can change it. The bus centralized the answer. A small integration team could change a routing rule or a transformation that every channel depended on, and internal control reporting would make that team's change records part of the financial controls. The outsourcing of mainframe operations in 2004 moved another set of hands outside the company, under a contract. **How fast it is fixed.** A fix on the bus reached every channel at once, which was its strength. It also waited in one team's queue, which was its weakness. **What AI changes.** An agent that can call services through a central gateway has the reach of the bus team without its change board. Applied to an agent, the 2000s answer would put its calls through the same published descriptions, logs and reconciliations as any other caller, with permissions narrower than the bus's.

DECISION CARD · DC-2007-01

2007 **Accepted** Owner: Head of enterprise architecture Supersedes: DC-1997-01

1 Decision	Connect the portal, the desk, the core and partners through one enterprise service bus with published service descriptions.
2 Outcome and binding constraint	One place to add a channel or a partner, and one governed path for payment instructions. **Binding constraint:** Integration cost and speed: each new channel needed its own links into the online transactions; financial flows must be controlled and auditable.
3 Boundaries and doors	**Portal and partners to bus**: published service descriptions and industry message standards **Bus to core**: the online transaction codes behind each bus service **Bus to payment gateway**: the payment instruction message
4 Options rejected	**More point-to-point links**: every new channel multiplied links to keep **Wait for the new core**: channels could not wait for a rewrite
5 Quality · Security · Visibility · Cost	**Quality.** Every service has a published description and a version. **Security.** The bus authenticates callers and logs every message into the core, but calls the core under one service identity, as the 1997 desk did. **Visibility.** Message logs per service; payment instructions reconciled with the bank file each day, ready for the internal control reporting coming for insurers. **Cost.** A central team and platform; each change queues for that team.
6 How it fails, and how we would see it	A changed message layout passes the bus and a field lands wrong in the core; or the central team becomes the queue for every change. **Evidence:** The daily payment reconciliation, and the age of the bus team's change queue.
7 Run	A central integration team; failed messages wait in the bus's store to be replayed.
8 Reversal cost	High: every channel and partner would need new links.
9 Assumptions and Revisit when	Most changes are new channels, not new core behavior; one team keeps pace. **Revisit when:** The bus team's queue is the slowest step in a change.
10 Change guide	1. No channel touches the core files; only the bus or the online transactions. 2. A published service description changes only with a new version.

Still Accepted in 2026: no later card replaces it, and what it still carries (portal inquiries, payment instructions, older partner feeds, one coverage lookup for an orphaned 2019-06 estimate deployment) follows it. Broker traffic, intake, the initial estimate and the reinsurer feed have moved off.

Monday rule: For every alarm you rely on, add a heartbeat from the alarm function itself, and send its failure to the people who act on the alarm, not only to the team that runs the server.

CHAPTER NINE

The 2010s: Many Teams, One Platform



On 25 March 2014 James Lewis and Martin Fowler published an article called "Microservices". A footnote recorded where the word came from: a workshop of software architects near Venice in May 2011, where it was "discussed", and the same group, which in May 2012 "decided on 'microservices' as the most appropriate name". Two years after the article, the insurer's claims department made its own version of the decision. It built claims intake as small services in containers that published events, beside a core that had run every night since 1974. This chapter is about why that was a sound answer to the decade's constraint, and about what it left unseen.

The constraint: many teams waiting for one release

By 2010 the scarce thing in a large software organization was rarely machine time or storage. It was the ability of many teams to change one system without waiting for each other. The 2000s had answered team delivery speed with agile methods inside a team. The 2010s met the next limit: dozens of agile teams still shared one release train, one database schema and one bus team.

The insurer felt it in a plain way. A change to how brokers reported a claim touched the portal, the bus, a CICS transaction and a copybook. Each had its own owner and its own release calendar. Mainframe operations had belonged to an outsourcer since 2004, so changes to the core also waited for the outsourcer's change window. A rule that a claims manager wanted on Monday shipped in the next quarter.

The decade's literature named the problem and offered a measure. Jez Humble and David Farley's *Continuous Delivery* (Addison-Wesley, 27 July 2010) argued that software should always be in a releasable state, with a deployment pipeline that every change passes through. Adam Wiggins's "The Twelve-Factor App", online by November 2011, set out rules for building services that a platform could run and replace without ceremony. Later in the decade, Nicole Forsgren, Jez Humble and Gene Kim's *Accelerate* (IT Revolution, 27 March

2018) gathered four years of State of DevOps survey research, begun in 2014, into a small set of measures. One of them, lead time for changes, is the time from a commit to that change running in production.

Conway had stated the underlying law in 1968: a system's structure copies the communication structure of the organization that builds it (Chapter Four). Lewis and Fowler cited it in their 2014 article ("This is an example of Conway's Law in action") and described the alternative they saw: "splitting up into services organized around business capability", each built by a cross-functional team.

Services, containers and a scheduler

Lewis and Fowler described microservices as a style, not a standard: a single application built "as a suite of small services, each running in its own process and communicating with lightweight mechanisms", each deployable on its own. The idea was older than the word. Subroutines (Chapter Three), modules with hidden decisions (Parnas 1972, Chapter Five) and the services of the 2000s all drew the same kind of line. What was new was cheap, repeatable deployment of many small units.

A few years of releases made that cheap. Docker's first public release, version 0.1.0, is dated 23 March 2013 in the project's change log. It packaged a process with its file system into an image that ran the same way on a laptop and a server. On 10 June 2014 Google announced Kubernetes as "a lean yet powerful open-source container manager"; version 1.0 was announced on 21 July 2015. The paper that explained Kubernetes's ancestry, "Large-scale cluster management at Google with Borg" by Abhishek Verma and colleagues, was presented at EuroSys in Bordeaux in April 2015, ten months after the Kubernetes announcement. Amazon previewed Lambda on 13 November 2014: functions that run in response to events, such as a file arriving in storage, with no server for the customer to manage.

The insurer adopted in its own order and at its own pace, as a mid-size mutual does. In 2013 it put a public broker API behind an API gateway beside the bus, so brokers' systems could file and query claims without screen scraping the portal. In 2016 it built claims intake and notifications as small services in containers that published events. This book does not name the platform that ran them: Kubernetes had reached version 1.0 only the summer before, and a mid-size insurer had several ways to run containers. The point of the decision was the boundary: intake could change weekly while the core, CICS and the bus kept their own calendars.

The 2013 gateway carried much of the decade's change. An API gateway sits in front of services and applies the same rules to every call: who the caller is, how many calls it may make, which version of the interface it asked for, and where the call goes. Before it, the rules

lived in each program or nowhere. After it, a broker's system met one published interface and one set of policies, whatever ran behind them. The gateway also became the place where an old version could be kept alive for a slow partner, which is a convenience and a risk.

Each service exposed an API described in a machine-readable form and published events with a defined schema. A broker's system, a notification service and an adjuster's screen could all rely on the same published shape, versioned and owned by one team.

The cost appeared in the run. A call that had been a branch inside one program became a call across a network, with the partial failures that Waldo and his colleagues described in 1994 (Chapter Seven). The insurer now had more deployments, more credentials and more dashboards. It needed people who could run containers and read traces. The services were smaller, but the system around them was larger. The same decade recorded the counter-case: on 10 July 2018 Segment's engineers described folding more than 140 services back into one ("It turns out, the solution for us was a monolith"), a case covered in depth later in this series.

The log as the spine

The log came out of a different problem. In June 2011 Jay Kreps, Neha Narkhede and Jun Rao of LinkedIn presented Kafka at the NetDB workshop in Athens as "a distributed messaging system that we developed for collecting and delivering high volumes of log data with low latency": activity events and operational metrics, not services. On 16 December 2013 Kreps published "The Log: What every software engineer should know about real-time data's unifying abstraction" on LinkedIn's engineering blog. His argument was simple: an append-only, ordered record of changes is the common structure under databases, replication and data integration. If every system writes its changes to a log, any other system can read them in order, at its own pace. In Kafka that order holds within a partition, not across a whole topic: related events, such as all the events of one claim, stay in order only if they share a key and so land in the same partition. That made the log a natural way for services that each own their data to tell others what changed.

Kreps took the idea from databases themselves: "Over-time the usage of the log grew from an implementation detail of ACID to a method for replicating data between databases." Change data capture (CDC), which his essay names as "the most log-friendly style of data extraction", uses that internal log to turn each committed change in a database into an event, without rewriting the programs that make the changes.

From 2016 the intake services published their events, such as "advance approved", on a log of their own at the edge. In 2018 the insurer added a firm-wide event log fed by CDC; intake also published its claim events there and kept its edge log. CDC captured changes from DB2 only. The DB2 tables held reporting data and a nightly copy of policy data. The open-claim

masters in IMS and the 1974 batch's VSAM files were not captured. The events that came out of CDC copied the old layouts field for field, and they described a copy, not the record. So any reader of the log saw policy data a day late and saw nothing of the open claims held in IMS. In November 2025 the deductible error of Chapter One reached the triage agents through this DB2 copy.

Since 2016 an intake rule, not a person, had approved advances below a set limit, and a payments clerk keyed each approved advance at the desk. In 2018 a bridge took the clerk's place: it read the "advance approved" events from the edge log and turned them into payment instructions on the bus, so a small advance could now travel from a broker's report to the bank with no screen in the path. Of the two 2018 additions, this unattended path carried the sharper risk: a message delivered twice can pay twice.

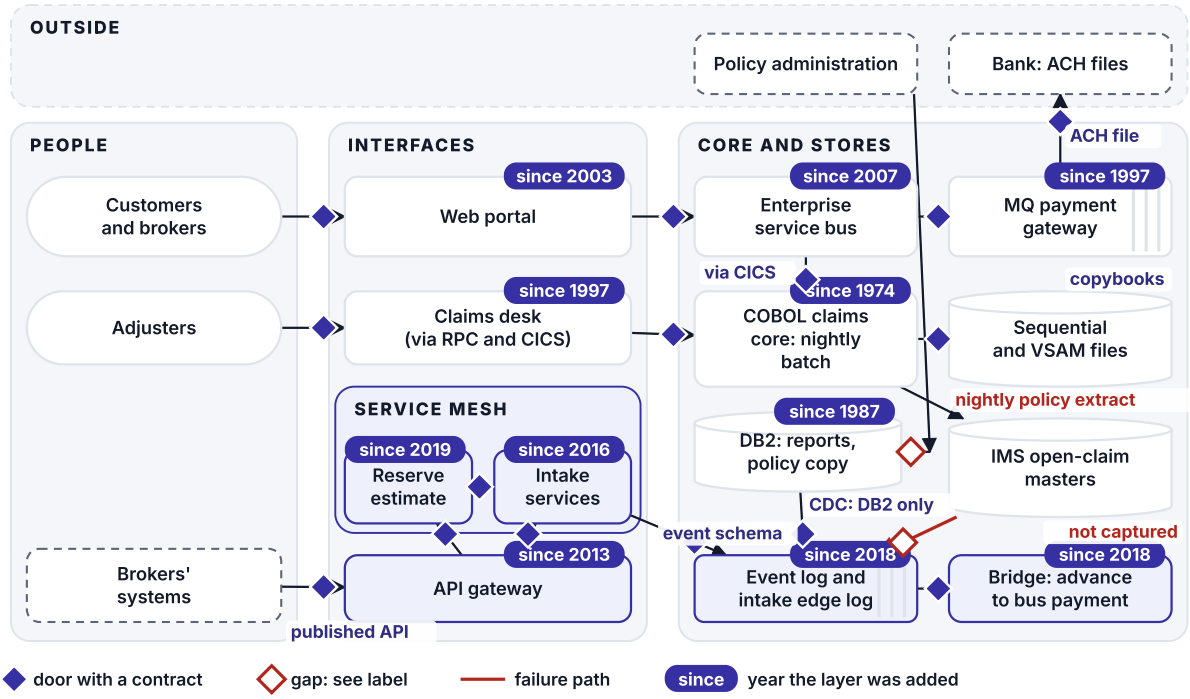
THE CLAIMS SYSTEM · 2018 · EVENT LOG AND CHANGE DATA CAPTURE FROM DB2

What was added	Changes to the DB2 data are captured and published as events on a log; IMS and VSAM are not captured, so the events copy the old layouts and miss the open-claim masters.
Door and its contract	The event schema of each captured change
In 2026	Still runs

In 2019 the initial reserve estimate at intake left the bus and became its own service, and a service mesh was added between the services. A mesh puts a small proxy beside each service. The proxies handle encryption between services, retries, timeouts and traffic rules, so each team does not write them again. Booked reserves stayed in the core's COBOL, where they had been since 1974.

FIGURE 9.1

2019: a gateway, services in a mesh, an event log and a bridge grow around a core the log sees only through a DB2 copy



The insurer's claims system in 2019; red gaps: the nightly extract of policy data from policy administration into DB2, and the absent feeds from IMS and VSAM, labeled not captured.

Author's diagram of the invented insurer; layer years are the insurer's adoption years.

Figure 9.1 shows the result on the canvas every decade figure in this book uses. The core and its files sit where they have sat since Chapter Five and still hold the claim of record. Every new layer of the decade sits around them: the gateway, the services in their mesh, the log and the bridge. The red doors mark the gaps the decade left, and Chapter Ten shows who inherited them.

WHAT HASN'T CHANGED

Strip away the vocabulary and the 2010s rebuilt the 1974 batch's habits in a new medium. The event schema is the copybook: a shared record layout that every reader depends on, and that no one may change alone. The log, ordered within each partition, is the sequential file that the nightly run read from top to bottom. The consumer that replays the log after a failure is the batch restarted from its last checkpoint. What changed was who could read the layout, and how many of them there were: in 1974 an incompatible change to a copybook meant recompiling every program that used it, in one shop; in 2018 a change to an event schema could break a consumer in a team the publisher had never met.

Running it: failure on purpose and the error budget

The decade's other change was in how organizations ran software. On 19 July 2011 Netflix described its "Simian Army" on its technology blog. Its best-known member, Chaos Monkey, "randomly disables our production instances", so that engineers build services that survive losing one. On 30 July 2012 Netflix released its source code; by default it ran on weekdays between 9 a.m. and 3 p.m., so that people were around to fix what broke. Instead of assuming that redundancy worked, Netflix tested it every working day.

Google's *Site Reliability Engineering*, whose foreword by Mark Burgess is dated March 2016, gave operations a vocabulary that development teams could share. A service level objective states how reliable a service must be. The gap between that objective and perfection is an error budget that a team may spend on change. When frequent misses spend the budget, in the book's words "releases are temporarily halted while additional resources are invested in system testing and development". The trade-off between speed and stability became a number both sides could see.

Regulation moved in the same decade. The EU's General Data Protection Regulation entered into force on 24 May 2016 and has applied since 25 May 2018. It made a firm answerable for where personal data went, which in a system of many services and a log means every copy, and it gave people a right to erasure (Article 17), which sits awkwardly with a log whose events are kept for a long time. Kafka's 2011 paper describes deleting messages after a retention period, "typically 7 days"; a firm that keeps events for months or years, to replay them, keeps every copy that long. For an insurer with EU policyholders, a 2018 log whose events copied personal data from DB2 was one more place to account for.

JUAN'S LENS

The decade drew good lines and then forgot to watch them. A service that reports healthy while its consumer silently drops events has not failed by its own measure, only by the business's. The test I apply to a 2010s layer is not "is each part up?" but "do the counts reconcile end to end?": events published against claims registered, instructions sent against payments made. If no one owns that reconciliation, the layer is not finished.

THE ERA'S SILENT FAILURE

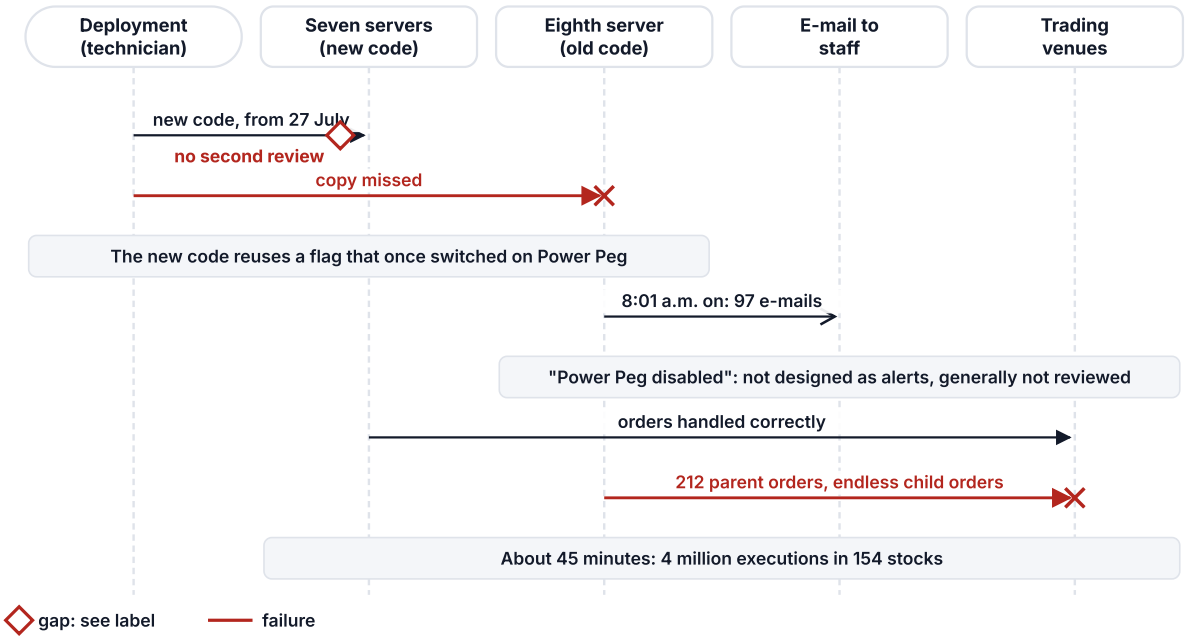
Knight Capital, 1 August 2012. That morning Knight Capital Americas LLC, a broker-dealer, sent millions of unintended orders into the US equity markets. The 45 minutes of trading were anything but quiet; the silent part came before them. The account here comes only from the SEC's order, Release No. 70694 of 16 October 2013.

Knight had written new code for a New York Stock Exchange Retail Liquidity Program. The new code "repurposed a flag that was formerly used to activate the Power Peg code", a function Knight had stopped using in 2003. In 2005 Knight had moved the function that counted executed shares to an earlier point in the code and "did not retest the Power Peg code", so the old code no longer knew when an order was filled (paragraphs 13, 14 and 16). From 27 July 2012 Knight deployed the code in stages. A technician "did not copy the new code to one of the eight SMARS computer servers"; there was no second technician's review and no written procedure that required one (paragraph 15). On 1 August, orders that carried the repurposed flag reached the eighth server and woke the old code. For 212 parent orders it kept sending child orders: "4 million executions in 154 stocks for more than 397 million shares in approximately 45 minutes" (paragraph 17). In one attempt to stop it, staff uninstalled the new code from the seven correct servers, which "worsened the problem" (paragraph 27). The SEC found that Knight lost more than $460 million.

Between the deployment and the open, nothing checked that all eight servers ran the same code. On the morning itself, from about 8:01 a.m., an internal system sent automated e-mails that "identified an error described as 'Power Peg disabled.'" It sent 97 of them to a group of Knight staff before the 9:30 a.m. open. The SEC records that Knight "did not design these types of messages to be system alerts" and that staff "generally did not review them" (paragraph 19). The SEC adds that the messages "provided Knight with a potential opportunity to identify and fix the coding issue prior to the market open". The signal went to a place no one watched, in a form no one treated as an alarm.

FIGURE 9.2

Knight Capital, 1 August 2012: 97 unreviewed e-mails before the open



From the SEC's findings: one of eight servers missed the new code, the repurposed flag woke the old Power Peg code, and automated e-mails that named the error went to staff before the open without being treated as alerts.

SEC, In the Matter of Knight Capital Americas LLC, Release No. 70694, 16 October 2013, paragraphs 13 to 19.

Knight's deployment was a manual copy to each server, the practice that the decade's deployment pipelines were built to replace. The SEC censured Knight, ordered it to cease and desist, imposed a civil penalty of $12 million and required an independent consultant to review its controls. It found violations of the market access rule, Rule 15c3-5, and of Rules 200(g) and 203(b) of Regulation SHO, because many of the unintended orders were short sales that went out marked as long, without the shares having been located to borrow (paragraphs 35 and 48).

SECURITY LENS

Who can change it. In the 2010s the answer moved from a change board to a pipeline. Anyone who can merge to a service's main branch can change production, and so can anyone who holds the pipeline's own credentials. Knight shows the other side: a manual copy by one technician, with no second review, left one of eight servers running old code. At the insurer, the 2018 bridge could move money with no one changing anything, and the gateway kept old interface versions alive for slow partners, each one more door to watch. **How fast it is fixed.** Continuous delivery made fixes fast as well as changes: a service that can ship ten times a day can patch ten times a day. A core on a quarterly outsourced window cannot, so the insurer's exposure in the core stayed measured in months while the edge's fell to hours. **What AI changes.** Little yet. Machine-learned code completion appeared late in the decade (Microsoft announced IntelliCode on 7 May 2018), but people wrote and reviewed the code. The decade did build the machinery the next one uses: pipelines that accept any commit that passes their tests, and logs that copy data to anyone who subscribes. Both assume the author is trusted. Chapter Ten asks what happens when it is not.

DECISION CARD · DC-2016-01

2016 **Superseded** Owner: Head of claims engineering Supersedes: none

1 Decision	Build claims intake and notifications as small services in containers that publish events, and leave the core, the bus and CICS as they are.
2 Outcome and binding constraint	A new intake channel or rule ships in days, not in the core's quarterly release, while the nightly batch still registers every claim. **Binding constraint:** Coordination between teams: every intake change queued behind the core and bus release calendar.
3 Boundaries and doors	**Portal and broker API to intake**: the published API description and its version **Intake to the rest of the firm**: an event schema per event type, such as advance approved **Intake to the core**: the CICS entry transaction, which stages an intake record
4 Options rejected	**Intake in the core's batch and CICS screens**: waits for the core and outsourcer calendars **One intake application beside the core**: all intake channels still share one release
5 Quality · Security · Visibility · Cost	**Quality.** Each service has its own tests and its own release; a schema change needs a version. **Security.** Each service gets only the credentials it needs; the gateway authenticates brokers. **Visibility.** Per-service health and latency dashboards; events counted in and out. **Cost.** More moving parts to run: containers, a log, many deployments, an on-call rota per team.
6 How it fails, and how we would see it	An event schema changes and a consumer drops events it cannot read, while every service reports healthy. **Evidence:** Events counted in and out of each service.
7 Run	Each team runs its own services; a platform team runs the containers and the edge log.
8 Reversal cost	Medium: the APIs and event schemas are costly to change; the services behind them can be rearranged.
9 Assumptions and Revisit when	Teams change intake often and independently; the network calls between services stay cheap and reliable. **Revisit when:** Each year: if changes keep spanning several services, or run cost grows faster than release speed.
10 Change guide	1. No event schema changes without a new version and a consumer check. 2. Intake never writes the core's files; it goes through the entry transaction.

Illustrative card for the invented insurer. Superseded for intake by DC-2024-01; notifications stayed a service, and the event schemas survived.

Monday rule: List every path added since 2010 on which money or a claim moves with no person watching, and for each one name the check that would stop it and who that check pages.

CHAPTER TEN

The 2020s (Our View): When AI Writes the Code



On 29 June 2021 GitHub announced a technical preview of Copilot, "a new AI pair programmer that helps you write better code". In the years that followed, models drafted code, tests and queries, and were then given tools to act on systems as well as to write for them. In September 2025 the insurer put AI agents into claims triage. This chapter is labeled our view because the decade is not over and its history is not settled. What we can describe is what became cheap: writing code. What that does to the binding constraint is our view, set out in the box near the end of the chapter.

The code you run includes code you did not write

The decade opened with two reminders that a system's boundary includes everything it depends on. On 13 December 2020 the US Cybersecurity and Infrastructure Security Agency ordered federal agencies, in Emergency Directive 21-01, to disconnect or power down SolarWinds Orion network management products that "are currently being exploited by malicious actors". A year later, on 10 December 2021, the record for CVE-2021-44228 was published: in Apache Log4j 2, a Java logging library, "an attacker who can control log messages or log message parameters can execute arbitrary code loaded from LDAP servers when message lookup substitution is enabled". Chapter Eleven returns to Log4Shell as a case where the logging machinery itself became the way in.

Both crossed a boundary that most architecture diagrams did not draw: the line between the code a firm wrote and the code it ran. A 2020s claims system runs open-source libraries, vendor agents, container base images and cloud services, each updated on its own schedule. The insurer's own count would include the mainframe software its outsourcer patches, the libraries inside its 2016 services and, from 2025, a model it does not train. The US government's response to the supply-chain side came in Executive Order 14028 of 12 May 2021, which called for guidance that includes "providing a purchaser a Software Bill of Materials (SBOM) for each product" (section 4(e)).

The 2010s had made change fast by trusting the pipeline: if a change passed the tests, it shipped. That trust assumed that the author of the change, and of everything the change pulled in, was known and accountable. The 2020s tested the assumption from two sides at once: from attackers inside the supply chain, and from a new kind of author.

When writing became cheap

GitHub made Copilot generally available on 21 June 2022. On 30 November 2022 OpenAI released ChatGPT. Within the decade's first half, a developer could ask for a function, a test suite, a migration script or an explanation of a COBOL paragraph and get an answer in seconds, which still had to be checked before it ran.

Measurements of the net effect are few and point in different directions. In a controlled experiment posted on 13 February 2023, developers with Copilot finished one set programming task 55.8% faster than a control group without it. In a randomized trial that METR published on 10 July 2025, 16 experienced open-source developers took 19% longer to complete issues when they were allowed to use early-2025 AI tools; beforehand they had expected AI to speed them up by 24%, and afterwards they still believed it had sped them up by 20%. In February 2026 METR marked those results as out of date. Its newer data pointed to a speedup with early-2026 tools, but developers who would not work without AI tended to stay out of the study, so METR called the data "only very weak evidence for the size of this increase".

Fashion swung in the same years, and one much-discussed architecture post was narrower than its fame. On 22 March 2023 Marcin Kolny of the Prime Video team published "Scaling up the Prime Video audio/video monitoring service and reducing costs by 90%". The post concerned one service, a tool that monitored the perceptual quality of every stream customers viewed. Its first version was a set of distributed serverless components, and the post names "the orchestration workflow" and the passing of data "between distributed components" as the two most expensive operations. The team moved it to "a monolith application" and reported: "Moving our service to a monolith reduced our infrastructure cost by over 90%." The same post said that the choice between microservices and a monolith "has to be made on a case-by-case basis". It was a measured result for one workload, not a verdict on microservices; the full case is covered in depth later in this series.

The insurer made a similar choice on its own evidence. Since 2016 claims intake had run as services, and since 2019 the initial reserve estimate at intake had run as a separate service inside a mesh. By 2024 the work had narrowed: almost every change touched both, and one small team ran the deployments, the mesh rules and the traces for what was, in practice, one capability. In 2024 the insurer merged intake and the reserve estimate back into one deployable, a modular monolith: one unit to build and release, with module lines enforced

inside it and the published API versions and event schemas unchanged outside it. Booked reserves stayed in the core. In 2023 the larger repair networks had begun sending webhooks into a cloud event router, another layer at the edge that this book only notes.

DECISION CARD · DC-2024-01

2024 **Accepted** Owner: Head of claims engineering Supersedes: DC-2016-01

1 Decision	Merge claims intake and the initial reserve estimate into one deployable, a modular monolith with enforced module lines, and keep the published API and event schemas unchanged.
2 Outcome and binding constraint	Intake and estimate changes ship together, checked by one test suite, with fewer cross-service failures and a lower run cost. **Binding constraint:** Verification and run cost: a small team tested and ran many services whose changes almost always moved together, across network failure modes. The 2016 choice, revisited.
3 Boundaries and doors	**Brokers and portal to intake**: the same published API versions as before **Intake module to estimate module**: a module interface checked at build time, not a network call **Monolith to the event log**: the same event schemas, versioned
4 Options rejected	**Keep the services and add more platform tooling**: it lowers toil but not the number of moving parts **Move intake into the core**: the core's release cycle is the constraint the 2016 decision escaped
5 Quality · Security · Visibility · Cost	**Quality.** Module boundaries enforced by build rules; one test suite; contract tests on the API and events. **Security.** One set of credentials for both modules, scoped to intake and estimate data: a wider reach than either service had, reviewed each release. **Visibility.** Traces per request inside the monolith; the same event counts in and out. **Cost.** Fewer deployments and less network traffic; one larger unit to scale.
6 How it fails, and how we would see it	Module lines erode until a change in one module silently alters another's output. **Evidence:** Build rules that fail on a forbidden dependency; contract tests on every published version.
7 Run	One team runs one deployable with a rollback per release.
8 Reversal cost	Medium: modules can be split out again because their interfaces were kept.
9 Assumptions and Revisit when	Intake and estimate change together and scale together. **Revisit when:** One module needs to scale or release on its own, or another team takes it over.
10 Change guide	1. Published API versions and event schemas do not change in the merge. 2. No module reads another module's tables.

Illustrative card for the invented insurer. Notifications, built as a service in 2016, are outside this decision.

The 2024 card supersedes the 2016 card; it does not reverse it. The 2016 decision drew the boundary that let intake change on its own schedule, and that boundary survived. What changed was the cost of running the boundary as a network.

Agents, tools and the claims queue

A model that only writes text can be wrong in a document. A model that can call tools can be wrong in a system. On 25 November 2024 Anthropic published the Model Context Protocol, describing it as "an open standard that enables developers to build secure, two-way

connections between their data sources and AI-powered tools". It gave agents a common way to discover and call tools, and it gave architects a new kind of interface to design: the tool, with its inputs, its limits and its owner. How agents plan and loop over tools is the subject of *The Agent Loop*; how many of them coordinate over events is the subject of *The Mesh*.

A tribunal had already said who answers for what a company's chatbot tells a customer. In *Moffatt v. Air Canada*, 2024 BCCRT 149, a small-claims decision of British Columbia's Civil Resolution Tribunal issued on 14 February 2024, an airline's website chatbot suggested to a customer that a bereavement fare could be applied for retroactively; the airline did not permit that. The airline argued that it could not be held liable for information given by the chatbot. The tribunal called this "a remarkable submission" and held: "It should be obvious to Air Canada that it is responsible for all the information on its website. It makes no difference whether the information comes from a static page or a chatbot" (paragraph 27). It found negligent misrepresentation and awarded 650.88 Canadian dollars in damages (paragraphs 32 and 40). The decision speaks only of a chatbot and does not say what technology ran it, and a small-claims decision binds no one else. Its point is narrow and plain: the company answered for what its website said, whatever produced the words.

The insurer designed its triage layer around the same point. From September 2025, AI agents propose a route for each new claim and an initial reserve estimate. They take intake from the 2024 modular monolith. They read the core in two ways only: through the 2018 event log, which carries DB2 changes and so sees policy changes a day late, and through one CICS policy-lookup transaction, the only same-day read. They propose estimates through the reserve estimate API, which the insurer publishes on its API gateway for brokers' systems as well, so the agents' calls meet the same version and rate rules; the policy check sits in front of that call. They never call the bus.

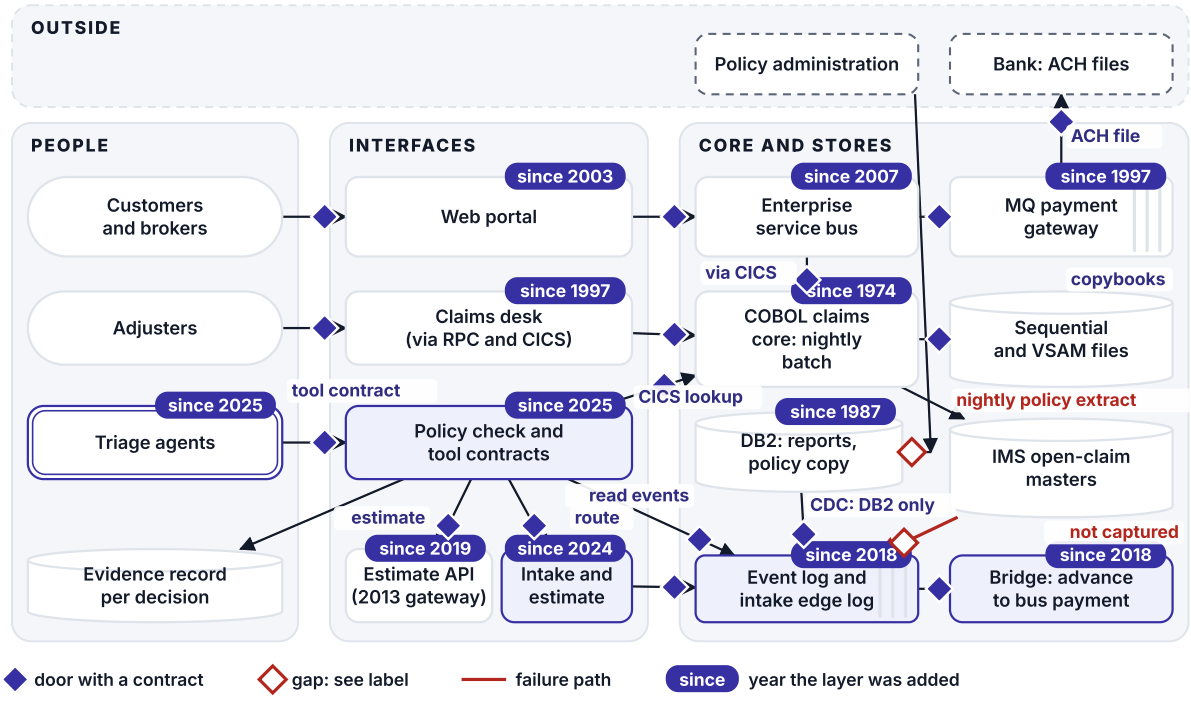
Every one of those calls passes through a policy check. Each tool has a written specification: the inputs it accepts, the values it may return, the limits it enforces, and who owns it. The check enforces the specification on every call and writes a record of the inputs, the proposal, the rule applied and the outcome. Two things have no tool at all. No agent can approve a payment: an intake rule approves advances below a set limit, as it has since 2016, and adjusters approve the rest. No agent can write booked reserves, which stay in the 1974 core. The first of these limits holds only if the rule reads nothing an agent produced. The insurer's rule reads the claimed amount, the policy's cover and the set limit, never a proposed route or estimate, so text that steers an agent cannot steer money. If a read path fails, triage falls back to rules only and claims keep flowing.

THE CLAIMS SYSTEM · 2025 · AI AGENTS TRIAGE CLAIMS UNDER CONTRACTS

What was added	AI agents triage claims under contracts and leave an evidence trail per decision (live September 2025). They read the core through the 2018 event log and one CICS policy-lookup transaction, take intake from the 2024 modular monolith, and only propose: no agent approves a payment or writes booked reserves.
Door and its contract	Tool contracts and the policy check in front of every action
In 2026	Still runs

FIGURE 10.1

2025: triage agents reach the claims system only through a policy check, with no path to payments or booked reserves



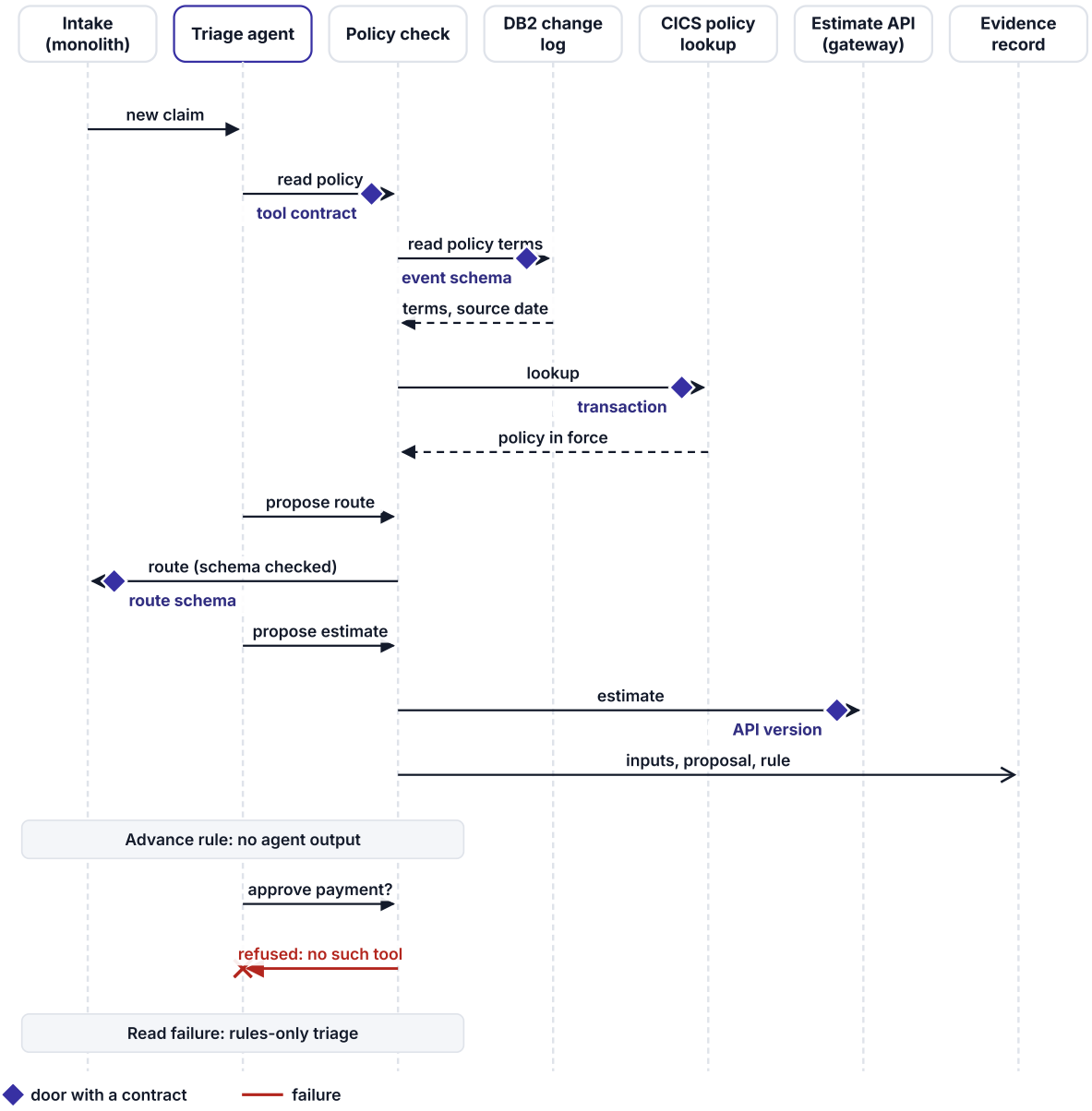
The insurer's claims system in 2025: the 2024 modular monolith and, from September 2025, triage agents that reach the system only through a recording policy check; the estimate API introduced in 2019 remains on the 2013 gateway, although its current implementation joined intake in 2024; the older 2019 deployment still serves the default broker route; red gaps: the nightly extract of policy data into DB2, and the absent feeds from IMS and VSAM, labeled not captured.

Author's diagram of the invented insurer; layer years are the insurer's adoption years.

Figure 10.1 places the agents on the same canvas as every earlier layer. They sit with the people, not in the core, and every line from them runs through one box. Its red doors are the gaps the agents inherit: the nightly policy extract into DB2, and the IMS and VSAM files the log does not see. Figure 10.2 follows one claim.

FIGURE 10.2

One claim through triage: the agent reads and proposes, the policy check routes or refuses, and each step is recorded



One claim through the insurer's triage layer since September 2025: the agent reads and proposes through tool contracts; the check routes or refuses and records each step; an intake rule that reads no agent output approves a small advance; if a read path fails, triage falls back to rules only. Policy terms come from the day-late DB2 change log; the CICS lookup checks only whether the policy is in force.

Author's diagram of the invented insurer.

The design is not new in its parts. A restricted interface in front of a powerful program is the insurer's 1982 CICS transaction. A record of every decision is the insurer's 1956 register with its control totals. What differs is the author behind the interface.

JUAN'S LENS

The model proposes; the contract decides. I do not ask whether a model is good enough to trust. I ask what it is allowed to do when it is wrong, and whether we would see it. For triage output that no person reviews, if the honest answer to the first question is "approve a payment" or "change a reserve", the design is not ready, however good the model. If the honest answer to the second is "a customer would call us", the design is not finished.

Regulation catches up

Rule-makers moved in the same years. Which of them binds a given insurer depends on where it writes business: the NAIC bulletin is a model for US states that choose to adopt it, while DORA and the AI Act are EU law. NIST released the AI Risk Management Framework 1.0 on 26 January 2023, "intended for voluntary use" to help bring trustworthiness into the "design, development, use, and evaluation" of AI products, services and systems. The National Association of Insurance Commissioners adopted its model bulletin, "Use of Artificial Intelligence Systems by Insurers", on 4 December 2023; it reminds insurers that decisions "made or supported by" AI systems must comply with insurance law, including the law on unfair trade practices and unfair discrimination, and tells them what documentation examiners may request. In the EU, the Digital Operational Resilience Act, Regulation (EU) 2022/2554 of 14 December 2022, has applied to financial entities, insurers among them, since 17 January 2025; it covers ICT risk, incident reporting and oversight of critical third-party providers. The EU AI Act, Regulation (EU) 2024/1689, entered into force on 1 August 2024 with obligations phased in from 2 February 2025; Regulation (EU) 2026/1744 of 8 July 2026 moved the application of the high-risk rules for the uses listed in the AI Act's Annex III to 2 December 2027, and for products under its Annex I to 2 August 2028. Whether a given use falls in those categories is a legal question to answer use by use, not a label to assume. The dates and their stages are in Appendix B.

Read as architecture, the documents overlap less than they seem to. DORA covers every ICT system, not AI in particular. On one narrower point, a way to stop an AI system, two of them meet. NIST's framework lists mechanisms to "supersede, disengage, or deactivate" AI systems that behave outside their intended use (MANAGE 2.4), and the AI Act requires that the people overseeing a high-risk system can halt it through "a 'stop' button or a similar procedure" (Article 14(4)(e)). Records of what a system did have an ancestor in earlier chapters: the 1974 batch's control report was a record of what ran, and who ran it.

OUR VIEW, AS OF NOVEMBER 2026

We read the decade's binding constraint as trust in AI-made work: verification. The ladder of constraints in Chapter Two has seen the pattern before: when one resource becomes cheap, the limit moves to the next. Producing a draft of a change became cheap; knowing that the change is correct did not. We think verification will stay the binding constraint for the rest of the decade. Models will keep getting better at producing code and decisions; the capacity of people, tests and production signals to confirm them will grow more slowly. Three consequences follow, as forecasts, not findings.

- Specifications will matter more than code. The written description of what a tool may do, and what an event means, will be the artifact teams review most carefully. It stays out of a model's reach only where its owner alone may approve a change to it, and repositories and pipelines enforce that rule.
- Evidence will move into the run. Reviews before release cannot keep up with machine-speed change; reconciliations, counts and sampled human checks while the system runs will carry more of the weight.
- Liability will shape architecture. After *Moffatt* and the NAIC bulletin, the design question "who owns this output?" will be answered in the system's structure, or it will be answered in court.

We may be wrong about speed. If automated verification, such as formal checks of generated code, matures faster than we expect, the constraint will move again, perhaps to energy or to cost.

THE ERA'S SILENT FAILURE

The xz backdoor, March 2024. On 29 March 2024 Andres Freund, a software engineer, wrote on the oss-security mailing list: "The upstream xz repository and the xz tarballs have been backdoored." The CVE record, CVE-2024-3094, states that "malicious code was discovered in the upstream tarballs of xz, starting with version 5.6.0". Both this case and the CrowdStrike outage below are told in full in *From Silicon to Screen*; this book keeps only the door each one crossed.

The door that failed was the one between a project's source and the archive that builds it. The bulk of the backdoor sat in the repository in obfuscated form, inside two binary test files; the step that unpacked it into the build was "solely in the distributed tarballs", the release archives, and not in the repository. A reader of the source saw test data, and no check compared what was built with what was reviewed. It was found through a side effect, not a check: on Debian sid, the distribution's development branch, logins with ssh were "taking a lot of CPU", and one engineer chose to find out why. No AI was involved. The case belongs here because it tests verification: the code that was built and run was not the code a reader of the repository saw.

For contrast, the CrowdStrike outage of 19 July 2024 was loud, but its door had been open, unseen, for five months. CrowdStrike's Root Cause Analysis of 6 August 2024 traces it to a template type shipped in February 2024 that "defined 21 input parameter fields, but the integration code ... supplied only 20 input values": two sides of one interface disagreed on a count, and, by the company's own findings, the count was checked neither when the sensor was built nor when it ran.

SECURITY LENS

Who can change it. In the 2020s the list includes the maintainers of every dependency, the vendor of every agent and model, and anyone whose text reaches a model's input. A claim description can carry instructions as well as facts: the attack NIST calls indirect prompt injection, possible because generative models "combine the data and instruction channels" (NIST AI 100-2, March 2025). The insurer's answer is that the agents hold no write tool for money or reserves, so a manipulated agent can propose a wrong route but cannot pay. **How fast it is fixed.** Fast where the boundary is known: the insurer can switch triage to rules only, and the claims flow does not depend on the agents. Slow where it is not: a backdoored library is fixed only after someone notices it, and xz was noticed through a side effect. **What AI changes.** Volume and fluency. More code and more decisions arrive, each well written. The insurer's answer is to limit what an unreviewed output can do, and to keep records that make a wrong one visible.

DECISION CARD · DC-2025-01

2025 **Accepted** Owner: Claims operations director Supersedes: none

1 Decision	Let AI agents triage claims only through tools whose specifications a policy check enforces on every call; agents read and propose, and the check and people decide.
2 Outcome and binding constraint	Every new claim gets a proposed route and an initial estimate within minutes, with a record of why. **Binding constraint:** Verification in surges: every new claim waited for a person to read it, and help with triage was worth having only if it could be checked as fast as it arrived.
3 Boundaries and doors	**Agent to policy check**: a tool contract per action: inputs, limits, allowed values **Policy check to the core**: read-only: the event log and one policy lookup transaction **Policy check to intake and estimate**: the route schema and the versioned estimate API
4 Options rejected	**Rules-only scoring, as before**: sends every unusual claim to one queue; kept as the fallback **Agents call the bus and core directly**: no single place to enforce limits or record decisions **Agents approve small payments**: payment approval stays with an intake rule and adjusters
5 Quality · Security · Visibility · Cost	**Quality.** A held-out set of past claims is replayed before any model or prompt change ships. **Security.** No write tool for payments or booked reserves; tool credentials scoped per tool. The advance rule reads the claimant's own claimed amount, so the set limit, not the agents, caps that exposure. **Visibility.** One evidence record per decision: inputs, proposal, rule applied, outcome. **Cost.** Model calls per claim plus the check; a budget per claim with an alert when it drifts.
6 How it fails, and how we would see it	A field renamed upstream makes the check fall back to a default route, and claims wait unseen. **Evidence:** Counts of claims in, routed and held, reconciled hourly; a monthly human sample of decisions.
7 Run	Claims operations owns outcomes; engineering runs the check; triage falls back to rules only if a read path fails.
8 Reversal cost	Low: switch triage to rules only; the claims flow does not depend on the agents.
9 Assumptions and Revisit when	Tool contracts can express the limits that matter; people review the sample. **Revisit when:** A new use of the agents is planned; a state's insurance rules or a court change what is expected of AI-supported decisions; or the sample shows drift.
10 Change guide	1. No tool may approve a payment or write booked reserves. 2. The advance rule never reads an agent's output.

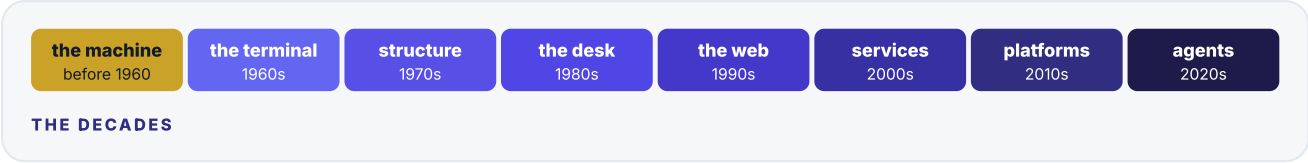
Illustrative card for the invented insurer.

Monday rule: For every AI agent or AI-written component in production, write down the one action it must never be able to take, and confirm that no tool, credential or code path lets it take that action.

PART III · WHAT CHANGED

CHAPTER ELEVEN

What Returned, and What Did Not Carry Over



On 4 June 1996, the first Ariane 5 flew normally for about 37 seconds and then broke up. The inquiry board found that software built for Ariane 4 had done exactly what its specification told it to do when it met an error: report it, store it and shut the processor down. The software, and the assumptions inside it, had been carried to a rocket whose flight they no longer fitted. This chapter is about that kind of carrying: the ideas that came back when the constraint moved, what stayed behind each time, and where the principles this book leads with failed or cost too much.

The same ideas, in new clothes

Read the decade chapters one after another and the same moves keep appearing. A unit of code that other code calls was a subroutine on EDSAC, a remote procedure in the 1980s and a service after 2000. A decision hidden behind an interface was Parnas's module in 1972, an object's interface in the 1990s and an API description in the 2010s. A record layout shared by many programs was a card layout, then a copybook, then a canonical message on a bus, then an event schema on a log. Figure 11.1 sets eight of these returns side by side.

FIGURE 11.1

What returned, and what did not carry over

Idea	First form	Its return	What did not carry over
Reuse a unit of code by calling it	The closed subroutine library (EDSAC, 1951)	The service and the remote call (RPC 1984; services from about 2000)	On one machine, caller and callee fail together; across a network either can fail alone, and the caller may not know whether the work was done (Waldo et al. 1994)
Hide a decision behind an interface	Information hiding (Parnas 1972)	Object interfaces (CORBA IDL, 1991), service descriptions, API descriptions	An interface does not make two components fit; their other assumptions must too (Garlan et al. 1995)
Share one record layout	Card layouts; COBOL copybooks	Canonical data models on the bus; event schemas on the log	The compiler applied a copybook to every program built with it; nothing applies a schema to every consumer
Run one big job in a window	The nightly batch	Stream processing and replay from a log	The batch's control totals; a stream needs its own reconciliation
Keep old programs running inside the new	Emulation on System/360 (1967 in the thread)	Wrapping, containers, the strangler fig (2004)	The wrap's cost: every layer keeps the shape of what it wraps
Put one team in charge of all change	The data processing department	The central bus team; then the platform team	The queue in front of that team became the constraint

Idea	First form	Its return	What did not carry over
Time-share one big machine	CTSS (1961) and the terminal	Multi-tenant cloud and shared clusters	Isolation that the owner of the machine could see and enforce
Let software write software	Compilers and generators (1950s)	AI that writes code (2020s)	Determinism: the same input no longer gives the same output

Each idea came back when the constraint moved, but something about its first setting stayed behind, and two of the failures this book tells, Ariane 501 and the Mars Climate Orbiter, sat in such a gap.

Author's table. Rows draw on: Wilkes, Wheeler and Gill 1951 (EDSAC subroutines); Birrell and Nelson 1984; Waldo et al. 1994; Parnas 1972; OMG CORBA history; Garlan et al. 1995; IBM Enterprise COBOL COPY statement; IBM System/360 Model 30 product profile; Fowler 2004; Saltzer on CTSS; Kreps 2013 (the log). The control-totals, team-queue and software-writes-software rows are the author's synthesis of Chapters 3, 8 and 10; the insurer's thread is illustrative. Full references in Appendix E, Chapter 11.

The fourth column matters most. An idea rarely returns whole. It returns into a new setting, and some property that its first setting gave it for free does not come along. A subroutine call on one machine fails together with its caller, so the caller never has to ask whether the other side is still there; across a network either side can fail alone, and the caller may not know whether the work was done. The compiler applied a copybook to every program built with it, though it could not tell whether every program had been rebuilt after a change; an event schema is applied only where a producer or a consumer chooses to validate against it. A nightly batch printed control totals that someone balanced; a stream has no end of night at which to balance anything.

Two of the silent failures in this book sat in that missing property: Ariane 501, told below, and the Mars Climate Orbiter. The orbiter's interface had a written specification, yet a file in the wrong unit crossed it and nothing at the boundary rejected it; when the navigators saw discrepancies, the team "relied on e-mail to solve problems, instead of formal problem resolution processes", the investigation board found (Chapter 7). The returns are easy to see. The properties that did not return are harder to see, and they are where the next failure tends to sit.

Why the old ideas return: Lehman's laws

The reason a system keeps its old ideas, and keeps adding new ones, was measured before most of this book's later decades began. L. A. Belady and M. M. Lehman measured the growth of IBM's OS/360 across its releases and published "A model of large program development" in the IBM Systems Journal in 1976, where they already postulated a law of continuing change. Lehman restated and extended the laws of program evolution in the Proceedings of the IEEE in September 1980.

The first two laws in that 1980 table describe the insurer's claims system closely. The law of continuing change: "A program that is used and that as an implementation of its specification reflects some other reality, undergoes continual change or becomes progressively less useful." The law of increasing complexity: "As an evolving program is continually changed, its complexity, reflecting deteriorating structure, increases unless work is done to maintain or reduce it."

The first law goes on to say that the change continues "until it is judged more cost effective to replace the system with a recreated version". Since the 1974 core arrived, the insurer judged so once, from 2006 to 2009, and abandoned the attempt. Every other year it chose the first law's alternative to decay: change, by adding a layer. Accretion is what the first law looks like when replacement keeps costing more than one more wrap. The second law is the bill for it, paid by whoever must understand every layer that still runs.

A contract is necessary, not sufficient

Several decade chapters showed designs that lasted because they drew a boundary and put an agreed, checked interface on it. The returns in Figure 11.1 show the limit of that observation. CORBA had an interface definition language, and web services had WSDL; both still met the problems below.

Two papers from the 1990s explain why. Jim Waldo and his colleagues at Sun Microsystems Laboratories wrote in November 1994 that "the hard problems in distributed computing concern dealing with partial failure and the lack of a central resource manager", and that "being robust in the face of partial failure requires some expression at the interface level." An interface that made a remote call look local hid exactly what the caller needed to know. David Garlan, Robert Allen and John Ockerbloom, in "Architectural mismatch: why reuse is so hard" (IEEE Software, 1995), showed that components with clean interfaces still failed to fit, because of assumptions the interfaces never stated: about the components themselves, the connectors between them, the structure of the whole and the way the system is built.

Both papers locate the trouble in the same place. An interface carries the promises its authors thought to write down; the failures they describe came from the promises left unwritten.

Where the book's principles failed or cost too much

Three of the series' principles lead in this volume: a checked agreement on every boundary, no silent failure, and evidence rather than assertion. Each has a documented case in which applying it, as written, cost more than it saved, or in which the machinery built to apply it opened a new risk.

No silent failure, against availability: Ariane 501. The full story of the flight is told in *From Tally to Token*; this book keeps only the boundary that failed. The inquiry board's report, dated 19 July 1996, traces the loss to the inertial reference system (SRI), whose design was "practically the same" as the SRI flying on Ariane 4 (section 2.1). An unprotected conversion "from 64-bit floating point to 16-bit signed integer value" overflowed in a function that, the report says, "serves no purpose" once the launcher lifts off (section 2.1). The specification then did what a fail-loud rule asks: on "any kind of exception", the failure was to be indicated on the databus, its context stored, "and finally, the SRI processor should be shut down" (section 2.2). The backup unit ran the same software and "had already ceased to function during the previous data cycle (72 milliseconds period)" (section 2.1), so redundancy gave no protection against a shared design error. The silent part sat at the next boundary: the active unit's output "showed a diagnostic bit pattern of the computer of the SRI 2, which was interpreted as flight data" (section 2.1). Nothing at the door between the SRI and the on-board computer told a failure report from a measurement, so the computer raised no error and steered on the pattern. The board recommended: "Do not allow any sensor, such as the inertial reference system, to stop sending best effort data" (section 4, R3). Judged by its era, the shutdown rule was sound for the faults it was written for; the board traced it to "the culture within the Ariane programme of only addressing random hardware failures" (section 2.2). Jim Gray's 1985 report (Chapter 6) argued that fail-fast modules run as process pairs "tolerate hardware faults and software Heisenbugs", the transient faults that do not recur on retry; a Bohrbug, a fault that fails again on retry, defeats the pair. Ariane's fault was of that second kind: the same input met the same code in both units.

A checked agreement on every boundary, against change: copybooks and XML schemas. A COBOL copybook is shared text that every program copies in when it is compiled. That made the record layout an agreement the compiler applied to every program built with it, and it made every change to that layout a change to every program that used it. The Year 2000 problem showed the cost. The US General Accounting Office's assessment guide (GAO/AIMD-10.1.14, September 1997) explains that systems "have typically used two

digits to represent the year ... in order to conserve on electronic data storage and reduce operating costs", that "the conversion of large databases from 2-digit to 4-digit year fields is a time consuming effort", and that testing and validation "may consume over half of the Year 2000 program resources and budget." Proving that every program and every file still agreed was the expensive part. The insurer's own remediation of its core ran from 1997 to 1999, through the copybooks and the programs, by windowing some dates and expanding others; a windowed date keeps a revisit date of its own, the end of its window. A decade later, strict schemas on web services raised the same problem in a new form. A draft finding of the W3C's Technical Architecture Group (13 November 2007), which states that it "does not yet represent the consensus opinion of the TAG", says: "Making versioning work in practice is one of the most difficult problems in computing." A W3C Working Draft of 20 July 2007, a guide to versioning with XML Schema 1.1, warned that open extension points "can become difficult to manage as the number of wildcards and definitions grows." In both cases, the tighter the agreement, the more each change cost.

Evidence, against security: Log4Shell. Logging is how a system keeps evidence of what it did. In December 2021 a feature of a widely used logging library became the way in. The US National Vulnerability Database record for CVE-2021-44228, published on 10 December 2021, reads: "An attacker who can control log messages or log message parameters can execute arbitrary code loaded from LDAP servers when message lookup substitution is enabled." It affected Apache Log4j 2 from version 2.0-beta9, and the Apache Logging project scores it 10.0, critical, its highest rating. The case is not a cost of keeping evidence as such; it is the cost of an evidence path built unsafely. Log4j's message lookups interpreted the strings it was asked to record, so text an attacker placed in a log message was treated as an instruction: the evidence path had become an input path.

> **JUAN'S LENS**
>
> I still hold all three principles, but each case taught me a qualifier I now write into the principle itself. Fail loudly, and keep serving what can still be served. Put an agreement on every boundary, and give it a version and a way to change. Keep evidence, and treat the evidence path as untrusted input. A principle without its qualifier is the rule that shut down the Ariane processor.

SECURITY LENS

Who can change it. Every return in Figure 11.1 moved the power to change something to a new party: from the compiler that applied a copybook to the team that owns a schema; from the operator who balanced the batch to whoever configures a stream; from a library's maintainers to every application that passes outside text into its log. **How fast it is fixed.** A shared component is fixed once and must be redeployed everywhere, and the first fix is not always the last. Log4j 2.15.0, the first fixed release, was itself "incomplete in certain non-default configurations" (CVE-2021-45046); the NVD records list 2.16.0, 2.17.0 and 2.17.1 as the Java 8 fixes for that flaw and two that followed it, each with releases for older Java versions. Finding every copy of the library inside every system took longer than any of the releases. **What AI changes.** AI that writes code returns one of the oldest ideas in the table, software that writes software, without determinism: the same request no longer produces the same output. What did not carry over is the ability to rerun the generator and get the same program, so the check has to move to the output.

Monday rule: When a team proposes a "new" pattern, ask which old one it returns, and name in writing the one property the old setting gave for free that the new one does not.

CHAPTER TWELVE

One Claims System Today: Keep, Wrap or Replace



On a night in November 2026, a broker files a water-damage claim on the insurer's web portal. Within minutes intake has taken the claim, an agent has proposed a route and, through the estimate API, an initial reserve estimate, a policy check has accepted both, and an adjuster has the file. Hours later, a COBOL batch first adopted in 1974 runs, as it does every night, and creates the claim of record. Before that claim is a day old, most of the layers of the last seventy years that still run have touched it. This chapter takes the system as it stands and asks the question every inheritor faces, layer by layer: keep it, wrap it or replace it.

The system as it stands

By November 2026 its claims system has added a layer at a time for seventy years; Figure 12.2 lists them. Three are gone: the punched-card shop of 1956, the 1401-class batch run of 1962 and the emulation that carried its programs from 1967. The rest still run, and two of them, intake and the initial reserve estimate, were merged into one deployable in 2024.

The shape is plain once it is listed. The 1974 COBOL core still creates the claim of record each night and still recalculates booked reserves, over sequential and VSAM files, with the open-claim masters in IMS. Since 2016, intake has handed each new claim to the core through the CICS entry transaction adopted in 1982, which stages it as an intake record for the nightly batch; intake never writes the core's files itself. CICS, adopted in 1982, still serves online inquiry and entry, and one of its transactions is the policy lookup the agents use. DB2, adopted in 1987, holds reporting data and a nightly copy of policy data; it is never the master. The 1991 RPC gateway still carries the claims desk's calls into CICS. The 1997 desk still runs, and so does the 1997 payment gateway, which reads payment requests from an MQ queue and turns them into files for the bank; since 2007 those instructions have reached it over the bus.

Above those sit the newer layers: the 2003 web portal; the 2007 enterprise service bus, now partly retired; the 2013 broker API behind an API gateway; services and events at the edge since 2016; the 2018 event log fed by change data capture from DB2 only; the estimate API introduced in 2019, whose current implementation joined intake in the 2024 deployable, while an older deployment remains behind the default broker route; the service mesh added between services the same year; and, since September 2025, AI agents that triage claims under contracts. The agents read around the core rather than into it: the event log, which carries DB2's copy of policy data, and one CICS policy-lookup transaction. They take intake from the 2024 deployable, and only propose. No agent approves a payment or writes booked reserves.

Figure 12.1 describes that system the way the series asks a platform to be compared: by what each capability must guarantee and how one would leave it, not by product. The products that offer these capabilities as cloud services are in Appendix D, dated, because their names change faster than the semantics.

FIGURE 12.1

One claims system today: the capabilities it needs, the semantics that matter, and the way out

As of November 2026. Product names change: check each vendor's documentation. No prices; cost shapes only.

Capability	Where the claims system uses it (2026)	Semantics to compare	Exit path
Batch run to a window	The 1974 core creates the claim of record each night and recalculates booked reserves	Runs to completion inside a window; restarts from a checkpoint; its output is the record of the day	Preserve record layouts and reconciliation during migration; creating the claim of record online is a separate decision
Short online transaction	CICS inquiry and entry; the policy lookup the agents use	Synchronous and short; a task limit caps concurrency; the screen map shapes the data	Wrap the transaction as an API with a published description (OpenAPI)
Durable queue	Payment requests to the payment gateway (1997), carried there by the bus since 2007	Persistent messages survive a restart; acknowledgement per message; a dead-letter path; a message redelivered after a failure can arrive twice, so the consumer ignores duplicates	Describe the channel and its message (AsyncAPI); keep the idempotency key in the message, not in the product
Central routing and transformatio n	The 2007 bus: portal inquiries, payment instructions on their way to the bank, the general ledger interface, older partner feeds, one policy lookup for the 2019 estimate deployment	Routing rules and mappings held in one place; one team changes them; a single path to watch	Service descriptions (WSDL) and industry payloads (ACORD); move routes off one at a time
API gateway	The broker API (2013) and the estimate API	Authentication, rate limits and routing by version at the edge; the default route matters	OpenAPI descriptions; route rules kept in version control
Ordered, replayable log	The 2018 event log: DB2 changes and edge events	Order per key, not globally; replay from an offset; retention by time or size	A product-neutral envelope (CloudEvents) and channel descriptions (AsyncAPI)
Change data capture	DB2 changes since 2018 (a nightly copy, so a day late); IMS and VSAM not captured	Committed changes in commit order; covers only the stores it can read; events copy the old layouts	An event schema per captured table, owned by a named team
Service-to-service traffic	The service mesh added in 2019 between services	Mutual TLS, retries and timeouts set outside the code; policy per route	Kubernetes Gateway API for routes; OpenTelemetry for what the mesh sees
Container runtime	Edge services and the 2024 modular monolith	Repeatable packaging on compatible hosts; platform dependencies tested; rollout and rollback per deployable	OCI images and Kubernetes manifests
Tracing and metrics	Every hop after 1997, where it can be instrumented	One trace across hops; sampling; who reads the alarm	OpenTelemetry (OTLP) to any back end
Agent tool calls under a policy check	The 2025 triage agents	Every tool call checked against its contract before it acts; agents propose only; one evidence record per decision	Tool descriptions in an open protocol (MCP); the policy kept outside the model vendor

Compare platforms on the semantics column and the exit path, not on product names. For the claims system two semantics matter most: duplicates on the payment path, which the consumer

discards by an idempotency key, and per-key order on the log. Appendix D maps the rows that clouds offer as services.

Author's table based on the illustrative claims-system thread; the standards named are published by their standards bodies.

The first wrap was in 1967

Wrapping is older than any word for it. IBM announced System/360 on 7 April 1964 as one compatible line of machines, and its product profiles for the Model 30 and Model 40 list a "Special 1401 compatible feature", so that programs written for the 1401 could keep running on the new hardware. In the thread, the insurer converted to System/360 in 1967 and kept its 1401 programs running unchanged under emulation. The old programs did their work as before; the new machine sat underneath them. That was the insurer's first wrap, a wrap from below rather than in front, and it bought seven years: the emulated programs ran until the COBOL core replaced them in 1974.

The pattern has repeated ever since, mostly in front. CICS in 1982 put screens in front of the core without changing how the core created a claim. The 1991 gateway put a remote interface in front of CICS. The 1997 desk put claim objects in front of the gateway. The 2007 bus put service descriptions in front of all of it. The 2018 log put events beside DB2. The 2025 agents read through the log and one transaction. Each layer wrapped the ones below it and left them running, which is how the 1974 core still registers every claim.

In the thread, each wrap was cheap when it was added, and its cost came later. Each wrap adds a boundary that someone must run, and each boundary inherits the shape of what it wraps. The 1997 claim objects were shaped by CICS screens; the 2018 events copy the layouts of the DB2 tables they capture. A wrap preserves the old system's assumptions as well as its behavior.

The strangler fig, and the replacement that stopped

Whole replacement has worked in the thread. The 1401 run replaced the card shop in 1962, and the COBOL core replaced the emulated 1401 programs in 1974; both times the old layer was small enough, and the new machine different enough, that a clean break was cheaper than a wrap. Since the core arrived, replacement has taken two forms, and only one has held. In 2006 the insurer started a rewrite of the claims core, all at once, and abandoned it in 2009. The bus it had added in 2007 was the part that survived. The core was not replaced; it was wrapped again.

The replacement that held went route by route. From 2013 brokers' systems called a public API on an API gateway beside the bus, not the bus itself. From 2016 intake moved to services. After 2018 the reinsurer feed moved to the event log. In 2019 the initial reserve

estimate became its own service. None of those moves switched the bus off; each took one route away, or kept a new one off it. By 2026 the bus still carries portal inquiries to the core, payment instructions on their way to the bank, the general ledger interface, older partner file feeds, and one policy lookup used by the 2019 version of the estimate service, an older deployment that still runs behind the default broker route. It is partly retired, which is what gradual replacement looks like from the inside.

Martin Fowler described that approach on 29 June 2004, in a post on his own site titled "Strangler Application", after the strangler fig, which, as he described it, seeds high in a host tree, works its way down to the soil and over many years strangles the host. He described the alternative to a rewrite this way: "An alternative route is to gradually create a new system around the edges of the old, letting it grow slowly over several years until the old system is strangled." He renamed the post "Strangler Fig Application" in April 2019 and replaced it with a longer version on 22 August 2024. The insurer's bus is being strangled in exactly that sense, slowly, one route at a time. The core has not been, because nothing yet creates the claim of record anywhere but in the nightly run.

Keep, wrap or replace, layer by layer

Figure 12.2 sets the seventy years out as a table. Read down the third column and the pattern is clear: most layers were kept, the oldest of them were wrapped again and again, and since 1974 the replacements that held were gradual.

FIGURE 12.2

Keep, wrap or replace: what seventy years did to each layer

Layer (insurer's adoption year)	What it still does in 2026	Treatment so far	What wraps or replaced it
Punched-card tabulation (1956)	Nothing; its card layouts survive as field positions	Replaced	The 1401-class batch run, 1962
1401-class batch run (1962)	Nothing	Wrapped, then replaced	System/360 emulation, 1967 (the first wrap); the COBOL core, 1974

Layer (insurer's adoption year)	What it still does in 2026	Treatment so far	What wraps or replaced it
COBOL claims core, VSAM and IMS (1974)	Creates the claim of record each night; recalculates booked reserves	Kept and wrapped many times; one replacement abandoned	CICS 1982, the RPC gateway 1991, the CORBA desk 1997, the bus 2007, the agents' lookup 2025; the 2006 to 2009 rewrite stopped
CICS inquiry and entry (1982)	Online inquiry and entry; the policy lookup the agents use	Kept and wrapped	The RPC gateway 1991; the bus 2007
DB2 beside the core (1987)	Reporting and a nightly copy of policy data	Kept	Its change log feeds the event log since 2018
RPC gateway (1991)	Carries the desk's calls into CICS	Kept	Nothing yet; the desk depends on it
Claims desk over CORBA, MQ payments (1997)	Claims handlers' desk; the payment gateway sends files to the bank	Kept; no longer the target design	The bus carries payment instructions since 2007
Web portal (2003)	Brokers and customers file and follow claims	Kept	Reaches the core through the bus
Enterprise service bus (2007)	Portal inquiries, payment instructions, the general ledger interface, older partner feeds, one policy lookup	Partly replaced, route by route	Broker API beside it 2013, intake services 2016, the reinsurer feed after 2018, the reserve estimate 2019
API gateway and broker API (2013)	Brokers' systems call the published API	Kept	
Services and events at the edge (2016)	Notifications; intake events	Partly merged back	Intake and reserve became one deployable, 2024

Layer (insurer's adoption year)	What it still does in 2026	Treatment so far	What wraps or replaced it
Event log and CDC (2018)	Publishes DB2 changes and edge events	Kept; IMS and VSAM still not captured; DB2 events arrive a day late	
Service mesh (2019)	Mutual TLS, retries and timeouts between services	Kept	
Agents under contracts (2025)	Propose routes; never pay, never book reserves	New	

Most layers were kept, and the oldest were wrapped again and again; the early whole replacements (1962, 1974) worked; since then replacement has held only route by route, and the one big-bang rewrite of the core was abandoned.

Author's table based on the illustrative claims-system thread used in this book (adoption years).

The insurer's position has a public parallel in government. The US Government Accountability Office reported in May 2016 that the federal government "spent about 75 percent of the total amount budgeted for information technology (IT) for fiscal year 2015 on operations and maintenance", and that agencies used systems with components that are, "in some cases, at least 50 years old" (GAO-16-468). Three of the book's six questions decide the next move for each layer.

- **What is its binding constraint now?** In 1974 the core was shaped by the cost of changing large programs and by the machine time a nightly window could buy, and a nightly batch was the right answer. Its constraint in 2026 is the night itself: a claim filed at 9 p.m. exists as a record only after the run. When volume rises, the window is what runs out.
- **Where are its boundaries, and which carry an agreement?** The copybooks are real agreements, with the rebuild gap Chapter 11 describes. The intake records that the entry transaction stages cross into the core each night: does anything balance them against the claims the batch created? The nightly extract into DB2 has no agreement at all. A read model is a copy of data shaped for reading. With capture from DB2 only, any read model built on the log misses the open-claim masters in IMS until that gap is closed.

- **What would show a silent failure?** For the core, the batch's own control totals still balance each night. For the newer layers, a dropped event or a default route can pass every health check, and an intake record the batch rejects is silent unless someone counts both sides. The log has a quieter gap: its policy events come from DB2's nightly copy, so they arrive a day late and in load order, and a consumer that treats them as live sees no error at all.

OUR VIEW, AS OF NOVEMBER 2026

Applied to the insurer, the answers point in different directions for different layers. The core is the hardest to replace and the most watched: keep it, and move its work off one function at a time. One fact outside the code weighs on that call: the mainframe has been operated by an outsourcer since 2004, under a contract that comes up for renewal in 2027, so any plan for the core is also a plan for who will run it. The bus is already being replaced gradually: finish the routes it still carries, one by one. The gateway, the desk and the payment path are kept because others depend on them, and each dependency is a boundary that needs an owner. The log is kept, and its gaps are named. How the insurer plans to move the claim of record, payments and booked reserves off the core is covered in depth later in this series.

WHAT HASN'T CHANGED

In 1967 and in 2026 the same question decided the wrap: can the new layer run beside the old one, unchanged, while work moves across? Emulation answered yes for the 1401 programs. The insurer's change data capture answers yes for DB2 and, as installed, no for IMS. Where the answer is no, a wrap of that kind, a copy running beside the old store, cannot start.

JUAN'S LENS

I would not ask whether the core is old. I would ask what it proves each night. Before I replace any layer, I want the new one to prove at least as much, at least as visibly, and I want that written on its Decision Card before the first route moves.

SECURITY LENS

Who can change it. Every wrap adds a party who can change behavior without touching the core: whoever edits the bus's routing rules, the gateway's default route, the 2019 mesh's route policies or an agent's tool contract. The core's copybooks change rarely and through a compiler; a route can change in minutes. List who can change each wrap, not only who can change the core. **How fast it is fixed.** The newer layers can be patched within hours; the core and its gateway move on a release calendar at the outsourcer, so a fix there crosses an agreement between two firms. **What AI changes.** AI that writes code now reaches the change path of the newer layers (Chapter 10), and the triage agents read the core's surroundings, not the claim of record: policy data a day late through the log, and one lookup that was built for people. That read path is a new party at an old boundary.

Monday rule: For each layer you inherited, write one line: keep, wrap or replace, the constraint that decides it, and the one thing the layer proves today that its successor must also prove.

CLOSING

Epilogue: Designed to Run, and the Next Binding Constraint

Every design in this book was judged twice: once by the constraint of its own decade, and once by how it ran afterwards. The 1974 claims core was a sound answer to the constraints of its decade. The book has traced two reasons it still runs: replacing it kept costing more than one more wrap (Chapters 11 and 12), and each night it shows, in totals someone checks, that it did what it was meant to do.

A design is judged by how it runs

A diagram shows what a system is meant to do. Running shows what it does. In the insurer's claims system, lasting and being visible are not the same thing. The core lasted and balances its totals each night; the nightly extract into DB2 and the bus also lasted. The extract checks a count, not the meaning of its fields. Payment traffic has daily reconciliation, but the bus has no reconciliation covering every route; logging a message does not prove its destination acted on it. What the thread does show is where the trouble came from: the layers that could fail and stay quiet, such as an extract with no agreed layout, a default route nobody watched, or an event dropped by a consumer that never said so.

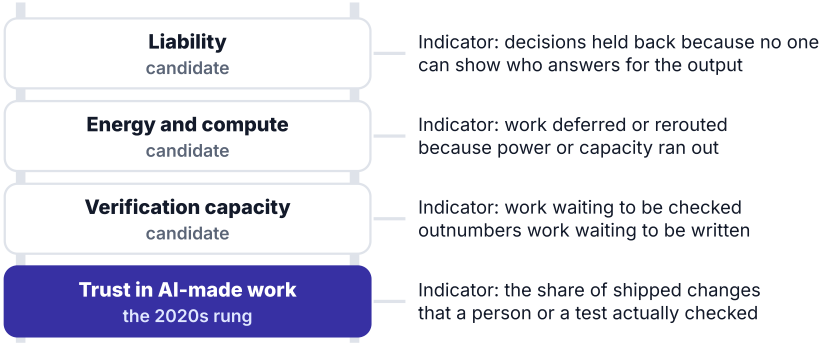
The series takes its name, *Every Boundary Holds*, from that observation, and its argument is a definition: a boundary holds when what crosses it is agreed and checked, and when a failure at it is seen. The decade chapters showed boundaries that held for fifty years and boundaries that failed in a single night. A written promise was not enough on its own: Ariane 501 and the Mars Climate Orbiter both had specifications (Chapters 7 and 11). The book's reading of its cases is that the boundaries that held also had someone arranged to notice when the promise broke. How to design a system to run, and to prove itself while it runs, is covered in depth later in this series.

What the constraint is now

Chapter 10, labeled as the book's view, named the binding constraint of the 2020s: trust in work that AI made. Code, tests, documents and decisions can now be produced faster than people can check them. The insurer's answer was to let agents propose and let contracts and people decide. That answer works while there are enough people, tests and checks to decide. It moves the constraint, as every answer in this book has, rather than removing it.

FIGURE EP.1

Candidates for the next binding constraint, each with the indicator that would show it binding



Our view only: the bottom rung is the 2020s constraint this book names; the three above it are candidates, each with the indicator that would show it binding. Verification capacity would tighten the current trust constraint rather than introduce a wholly new one.

Author's view, as of November 2026.

OUR VIEW, AS OF NOVEMBER 2026

Three candidates could become the next binding constraint. We hold each loosely, and we name the indicator that would show it binding, because the book's own method asks for that.

- **Verification capacity.** This is a tightening of the current constraint, not a wholly new one: checks worth trusting exist, but there are too few of them for the volume. If producing a change becomes nearly free, the scarce resource is the ability to show that the change is right: tests worth trusting, reviewers who understand the domain, contracts that machines can check. The indicator is the queue: work waiting to be checked rather than work waiting to be written.
- **Energy and compute.** The International Energy Agency's 2026 report *Key Questions on Energy and AI* gives updated projections of electricity consumption from data centers "roughly doubling from 485 TWh in 2025 to 950 TWh in 2030". If that growth meets limits on power or capacity, the cost of running models becomes the constraint on what is built, as machine time was before 1960. The indicator is work deferred or rerouted because capacity ran out.
- **Liability.** The EU's revised product liability directive, Directive (EU) 2024/2853, extends the definition of a product to software. Member states must transpose it by 9 December 2026, and it applies to products placed on the market or put into service after that date. It covers damage to natural persons: death or personal injury; damage to property, except the defective product itself and "property used exclusively for professional purposes"; and "destruction or corruption of data that are not used for professional purposes" (Article 6). It does not apply to free and open-source software supplied outside a commercial activity (Article 2). So it does not reach an automated decision whose only harm is financial, whether to a business or to a consumer whose claim is routed wrongly or late, and that gap is the point. If the answer to "who is responsible for this output" stays unclear, organizations will hold back decisions they could automate. The indicator is decisions waiting for someone to sign for them.

These constraints stack, as earlier ones did. Machine cost came back as cloud cost; the application backlog came back as the review backlog. The next decade will probably be bound by more than one of them at once.

SECURITY LENS

Who can change it. Each candidate widens the set of parties who can change a system without touching its code: whoever can publish a release of a dependency, as the xz backdoor showed (Chapter 10), and whoever can reach a flaw in one, as Log4Shell showed (Chapter 11); the provider of a model; whoever sets the policy an agent is checked against. **How fast it is fixed.** If verification is the constraint, security fixes wait in the same queue as every other change, so the queue needs a lane for them. **What AI changes.** A change proposed by a model and accepted by a check is only as safe as the check; the evidence of who proposed and who accepted is what an investigator will ask for first.

The six questions, again

The book began with six questions and has asked them of every decade. They work on the 1956 card shop and on the 2025 agents. They apply to the next design that reaches a team, whoever or whatever drafted it.

1. What outcome does it serve?
2. Where are its boundaries?
3. Who can change it?
4. How does it fail, and what would show us?
5. What does it cost to run and to change?
6. What assumption, if it changed, would make it wrong?

The sixth question is the one history answers most often. Every decade's design was right until its assumption moved: machine time became cheap, programs grew past one person, users escaped the data processing department, teams outnumbered machines. An assumption that is written down can be watched; that is why the Decision Card gives it a field and a revisit trigger.

The insurer's 1974 batch will run again tonight. It will create the claim of record for every claim filed today, including the ones an agent triaged in minutes. Somebody will read its totals in the morning. That morning check is the test this book has put to every layer since 1956.

APPENDICES

The Decision Card: Blank and Worked

The Decision Card records one architectural decision on one page: what it decides, what it serves, where its boundaries are, how it fails and when to reopen it. This appendix gives the blank card, a guide to each field, an index of the worked cards in this book, and the life of one decision across fifty-two years.

The blank card

Copy it, print it or keep it beside a design document. A card is finished when someone who was not in the room, a new engineer or an agent, could identify its constraints, required checks and supporting design records without having been in the room.

DECISION CARD · BLANK

Decision id Date Status: Proposed | Accepted | Rejected | Superseded | Reversed | Retired

Owner (a job title) Supersedes

1 Decision One sentence.	
2 Outcome and binding constraint What it serves; what is running out.	
3 Boundaries and doors What crosses where, and the contract on each door.	
4 Options rejected And why.	
5 Quality · Security · Visibility · Cost One line each.	
6 How it fails, and how we would see it The silent failure named; the evidence that shows it.	
7 Run Who operates it, how it recovers, the shape of its run cost.	
8 Reversal cost What undoing it takes.	
9 Assumptions and Revisit when What must stay true, and the trigger to reopen it.	
10 Change guide What any changer, person or agent, must not break.	

The six questions

1. What outcome does it serve?
2. Where are its boundaries?
3. Who can change it?
4. How does it fail, and what would show us?
5. What does it cost to run and to change?
6. What assumption, if it changed, would make it wrong?

The header

- **Decision id.** The letters DC, the year and a two-digit number, joined by hyphens: DC-1974-01. The id never changes; a changed decision gets a new card, which names this one in Supersedes.
- **Date.** The date the decision took effect, not the date the card was written. A card written later about an old decision says so in its note.

- **Owner.** A job title, never a person's name: people move, roles stay. If no role owns the decision, that is the first finding.
- **Supersedes.** The id of the card this one replaces, if any. The chain of supersedes links is the decision's history.

Status takes one of six values, and their meanings are narrower than everyday use.

Status	Meaning	What it does not mean
Proposed	Written and under review; nothing has been built on it yet	That it is likely to be accepted
Rejected	Considered and not chosen; the reasons remain in the record	A decision put into effect and later undone; that is Reversed
Accepted	In force: the system is built and run on this decision today	That it is still the best choice; only that it governs
Superseded	No longer the target design, usually because its constraint moved: a later card names what new work should follow	That it was switched off. The 1997 claims desk (DC-1997-01) is superseded by the 2007 bus card (DC-2007-01) and still runs every day
Reversed	Undone after it took effect: work stopped and what was built was withdrawn or abandoned	That it was wrong for its time; field 2 records the constraint it answered and field 4 what it was weighed against
Retired	The decision and what it built are gone from the running system	That nothing of it survives; layouts and habits often do

The ten fields

1. **Decision.** One sentence, with a verb. "Use an event log" is a topic; "Publish every committed change to a claim as an event, in commit order" is a decision.
2. **Outcome and binding constraint.** The business outcome the decision serves, and the thing that was running out when it was made. This is the field a later reader needs most, because it is how they judge the decision by its own era before judging it by theirs.
3. **Boundaries and doors.** What crosses where, and the agreement on each crossing: a record layout, a transaction code, an interface description, an event schema, a tool's contract. Mark any crossing with no agreement. Three at most on the card; the rest go in the design document.

4. **Options rejected.** The serious alternatives and why each lost, in the terms of field 2. Include buying when building was chosen, and the reverse.
5. **Quality, Security, Visibility, Cost.** One line each: what the decision guarantees, who can change what it protects, how anyone would see it working, and the shape of what it costs.
6. **How it fails, and how we would see it.** Name the silent failure, the one that passes every check, and the signal that would show it. If there is none, say so: that is a finding.
7. **Run.** Who operates it, how it recovers, and the shape of its run cost. A decision nobody can run is not finished.
8. **Reversal cost.** What undoing it would take: time, money, data moved, contracts reopened. A high reversal cost is not a reason to refuse a decision; it is a reason to write the card carefully.
9. **Assumptions and Revisit when.** What must stay true for the decision to stay right, and the event that should reopen it. The best revisit triggers can be observed: a volume, a date, a contract renewal.
10. **Change guide.** The invariants any changer, person or agent, must not break. This is the field an agent reads before it proposes a change, so write it as rules a machine could check.

The worked cards in this book

Each decade chapter has at least one worked card, and the 1960s, 2000s and 2020s have two. Each is written for a decision that the illustrative insurer, like real companies of the era, would have faced, judged by that decade's constraint. The cards are retrospective: written now, as worked examples, about decisions dated then. Together they follow the claims system through its life. The early cards are Retired because what they built is gone; the 2007 bus card stays Accepted while the bus is partly retired, because it still governs every route the bus carries, and it will be superseded when the last route moves.

The worked Decision Cards in this book

Id	Date	Status	Decision
DC-1956-01	1956	Retired	Punch every claim on cards in a fixed layout and tabulate claims in batches, each released only when its control totals agree.
DC-1962-01	1962	Retired	Move the claims run from the card shop to a stored-program batch computer, keeping the card layouts as its input.
DC-1967-01	1967	Retired	Convert to the supplier's new compatible machine family and run the old claims programs unchanged under emulation until they are rewritten.
DC-1974-01	1974	Accepted	Create the claim of record and recalculate booked reserves in one nightly batch over shared record layouts, with the open-claim masters in a hierarchical database.
DC-1982-01	1982	Accepted	Put claim inquiry and entry online through a transaction monitor over the core's records, and keep the claim of record in the nightly batch.

Id	Date	Status	Decision
DC-1997-01	1997	Superseded	Wrap the core behind claim objects on an application server for a three-tier desk, and reach the online transactions through the 1991 remote-call gateway.
DC-2006-01	2006	Reversed	Replace the claims core in one big-bang program with a new system that creates the claim of record online.
DC-2007-01	2007	Accepted	Connect the portal, the desk, the core and partners through one enterprise service bus with published service descriptions.
DC-2016-01	2016	Superseded	Build claims intake and notifications as small services in containers that publish events, and leave the core, the bus and CICS as they are.
DC-2024-01	2024	Accepted	Merge claims intake and the initial reserve estimate into one deployable, a modular monolith with enforced module lines, and keep the published API and event schemas unchanged.

Id	Date	Status	Decision
DC-2025-01	2025	Accepted	Let AI agents triage claims only through tools whose specifications a policy check enforces on every call; agents read and propose, and the check and people decide.

The life of one decision: DC-1974-01

Card DC-1974-01 (Chapter 5) records the decision to create the claim of record and recalculate booked reserves in one nightly batch over shared record layouts. The insurer built it in COBOL, with the open-claim masters in IMS; those names belong to the history in the table below, not to the card. Its status has never changed: Accepted, because it still governs, even though its Run owner changed in 2004 and Chapter 12 weighs moving its work off. Everything around it has changed.

DC-1974-01, from 1974 to 2026 (the insurer's adoption years)

Year	What happened around the decision	Card
1974	Accepted. The nightly COBOL batch creates the claim of record; the 1967 emulation of the 1401 programs is retired	DC-1974-01
1982	CICS online inquiry and entry are put over the core's files; the claim of record stays in the batch	DC-1982-01
1987	DB2 is put beside the core for reporting and a nightly copy of policy data; the core keeps its files as the master	
1991	Branch PCs reach the CICS transactions through an RPC gateway; the core is unchanged	

Year	What happened around the decision	Card
1997	Wrapped: the three-tier desk exposes claim objects over CORBA and reaches CICS through the 1991 gateway; payments move to EFT through an MQ queue	DC-1997-01
1997 to 1999	Y2K remediation: two-digit years found and fixed through the copybooks and the programs	
2004	Mainframe operations move to an outsourcer. The card's Run field changes owner, which is a revisit trigger: the decision stands, but who can run and change it does not	
2006 to 2009	A rewrite of the core is started and abandoned in 2009; the 2007 bus survives it	DC-2006-01 (Reversed)
2007	The bus wraps the core for the portal, the desk and partners, and from 2007 carries the payment instructions to the payment gateway	DC-2007-01
2016	Intake moves to services and hands each new claim to the core through the 1982 entry transaction, which stages it as an intake record for the nightly batch; intake never writes the core's files, and the batch still creates the claim of record	DC-2016-01
2018	Change data capture from DB2 only; the core's IMS and VSAM files are not captured	

Year	What happened around the decision	Card
2024	Intake and the initial reserve estimate become one deployable; it still hands each claim to the entry transaction, and nothing else changes at the core's boundary	DC-2024-01
2025	Agents read around the core, through the event log and one CICS policy-lookup transaction, and only propose	DC-2025-01
2026	Still Accepted: the batch still registers every claim of record each night and recalculates booked reserves	DC-1974-01

Read down the middle column and the card's value shows. Every later card had to work around this one. A team with DC-1974-01's change guide in hand, in 1997 or in 2025, would have known what the new layer must not break: the claim of record is created in the nightly cycle, and nothing else writes booked reserves.

Writing your first card

- Start with a decision already in force, not a new one. Writing down what a system already assumes finds more than designing on paper.
- Fill field 2 and field 6 first. If you cannot name the constraint or the silent failure, the decision is not yet understood.
- Keep product names off the card. The table above names products because it is history; a card states the decision, and the products go in a dated mapping, like Appendix D, so the card does not go stale when a product is renamed.
- Review the card when its revisit trigger fires, not on a calendar.

Timeline, 1945 to 2026

The dated landmarks the chapters treat, in order. Where a work was published in one year and carries another on its imprint, or was announced before it was available, both dates are given. The last column shows the insurer's layers, which are illustrative: the year there is the year the insurer adopted the idea, usually years after its release.

The last column is aligned by date only: a landmark and the insurer's layer in the same row are related only where they name the same idea. Where they do, the row shows the gap between the leading edge and the installed base. IBM first shipped CICS, as PU-CICS, in 1968; the insurer put claims online on it in 1982. DB2 was announced in 1983; the insurer put it beside its core in 1987. The chapter column says where each landmark is told, and Appendix E gives its source. A few entries before 1945 are kept because the chapters build on them.

Before 1960

Year	Landmark	Dates	Ch	The insurer adopts
1890	Hollerith tabulating machines count the US census	1890	3	
1931	(the insurer's own row)		3	Policy records on punched cards (the separate policy system)
1945	First Draft of a Report on the EDVAC	Dated 30 June 1945	3	
1949	EDSAC runs its first logged program	6 May 1949	3	
1949	IBM 407 accounting machine introduced	1949 (first manual of information)	3	

Year	Landmark	Dates	Ch	The insurer adopts
1951	Wilkes, Wheeler and Gill, the subroutine library book	Addison-Wesley, 1951	3	
1951	LEO I's first regular business job	Test run 5 September; regular run 28 to 30 November 1951	3	
1952	Wheeler, "The use of sub-routines in programmes"	ACM national meeting, May 1952	3	
1952	Society of Actuaries committee report on recording and computing devices	September 1952	3	
1953	IBM 650 announced	Development announced 14 July 1953 (IBM's date; some histories give 2 July); first delivery December 1954	3	
1954	Metropolitan Life installs UNIVAC	1954	3	
1954	Luhn files his check-digit patent	Filed 6 January 1954; granted 23 August 1960	3	
1956	Canning, Electronic Data Processing for Business and Industry	1956	3	1956: punched-card claims tabulation

Year	Landmark	Dates	Ch	The insurer adopts
1957	FORTRAN paper; compiler distributed	Paper February 1957; distributed April 1957	3	
1959	COBOL meeting at the Pentagon	28 and 29 May 1959; specifications published April 1960	3	
1959	IBM 1401 announced	Announced 5 October 1959; first shipped September 1960	4	1962: stored-program batch run

The 1960s

Year	Landmark	Dates	Ch	The insurer adopts
1961	CTSS demonstrated at MIT	November 1961; paper May 1962	4	
1962	Mariner 1 lost	22 July 1962	4	
1964	IBM System/360 announced	7 April 1964; Models 30 and 40 with a 1401 compatibility feature	4	1967: conversion with 1401 emulation
1964	First BASIC programs run at Dartmouth	1 May 1964	4	
1964	SABRE takes over American Airlines reservations	First tested 1960; took over about four years later	4	
1968	Dijkstra's "Go To" letter	CACM, March 1968	4	

Year	Landmark	Dates	Ch	The insurer adopts
1968	Conway, "How Do Committees Invent?"	Datamation, April 1968	4	
1968	IMS first runs at Rockwell	First "READY" at Rockwell, 14 August 1968; announced by IBM in 1968; released as IMS/360 in 1969	4	1974: open-claim masters in IMS
1968	CICS first shipped	First shipped as PU-CICS, free code for public utilities, in 1968; the CICS program product followed on 8 July 1969	4, 6	1982: online inquiry and entry
1968	NATO conference at Garmisch: the "software crisis"	7 to 11 October 1968; report January 1969	4	
1969	IBM unbundles software from hardware	IBM bulletin, 23 June 1969	2	
1969	NATO conference in Rome	27 to 31 October 1969; report April 1970	4	

The 1970s

Year	Landmark	Dates	Ch	The insurer adopts
1970	Codd, the relational model	CACM, June 1970	5	
1970	Kunz and Rittel, Issue-Based Information Systems	Working paper, July 1970	1	
1970	System/370 announced	30 June 1970	5	
1970	Royce, managing the development of large software systems	WESCON, August 1970	5	
1972	VSAM first planning guide	First edition July 1972	5	
1972	Parnas, decomposing systems into modules	CACM, December 1972	5, 11	
1973	Equity Funding: trading halted	March 1973	5	
1974	Liskov and Zilles, abstract data types	SIGPLAN Notices, April 1974	5	
1974	Stevens, Myers and Constantine, "Structured Design"	IBM Systems Journal, 1974	5	
1974	(the insurer's own row)		5	1974: COBOL claims core, nightly batch, VSAM files, open-claim masters in IMS

Year	Landmark	Dates	Ch	The insurer adopts
1974	Tandem Computers founded	Established November 1974; first NonStop system delivered May 1976	5	
1975	Brooks, The Mythical Man-Month	1975	5	
1975	Saltzer and Schroeder, protection of information	Proceedings of the IEEE, September 1975	5	
1976	Metcalfe and Boggs, Ethernet	CACM, July 1976	6	
1976	Belady and Lehman, a model of large program development	IBM Systems Journal, 1976	5, 11	
1978	Gray, notes on database operating systems	1978	5	
1979	NORAD false alert from test data	9 November 1979; further alerts 3 and 6 June 1980	5	

The 1980s

Year	Landmark	Dates	Ch	The insurer adopts
1980	Lehman, laws of program evolution	Proceedings of the IEEE, September 1980	5, 11	
1981	IBM PC unveiled	12 August 1981	6	

Year	Landmark	Dates	Ch	The insurer adopts
1981	RFC 801 plans the switch to TCP/IP	November 1981; switch by 1 January 1983	6	
1983	Lotus 1-2-3 shown to user groups	Presented at the Boston Computer Society, 23 February 1983	6	
1983	DB2 announced	Announced 7 June 1983; version 1.1 on MVS 1985	6	1987: DB2 beside the core
1983	IEEE 802.3 approved	23 June 1983	6	
1984	Birrell and Nelson, remote procedure calls	ACM TOCS, February 1984	6	1991: RPC gateway for branch PCs
1984	Goldratt, The Goal	First published 1984	Brief	
1984	Saltzer, Reed and Clark, end-to-end arguments	ACM TOCS, November 1984	6	
1985	Gray, "Why Do Computers Stop"	Tandem report, June 1985	6	
1985	Therac-25 accidents	June 1985 to January 1987	6	
1986	Brooks, "No Silver Bullet"	IFIP 1986; IEEE Computer, April 1987	6	
1989	Berners-Lee's proposal for the web	March 1989	7	

Year	Landmark	Dates	Ch	The insurer adopts
1989	Gabriel, "worse is better"	Invitation 1989; talk March 1990; AI Expert June 1991	6	

The 1990s

Year	Landmark	Dates	Ch	The insurer adopts
1991	CORBA 1.0 with IDL	October 1991 in OMG's history; August 1991 in its specification catalog	7	
1992	Perry and Wolf, foundations of software architecture	October 1992	1	
1993	CERN puts the web software in the public domain	30 April 1993	7	
1993	MQSeries introduced	1993	7	1997: MQ payment gateway
1994	Waldo et al., "A Note on Distributed Computing"	November 1994	7, 11	
1994	Gamma, Helm, Johnson and Vlissides, Design Patterns	Published 31 October 1994; imprint 1995	7	
1995	Sun announces Java	23 May 1995; Java 1.0 available 23 January 1996	7	

Year	Landmark	Dates	Ch	The insurer adopts
1995	CORBA 2.0 with IIOP	Specification dated July 1995, updated July 1996; OMG's history lists August 1996	7	1997: three-tier desk over CORBA IIOP
1995	Garlan, Allen and Ockerbloom, architectural mismatch	IEEE Software, 1995	11	
1995	Kruchten, the 4+1 view model	IEEE Software, November 1995	1, 7	
1996	Ariane 501 lost; inquiry report	Flight 4 June 1996; report 19 July 1996	11	
1997	GAO Year 2000 assessment guide	September 1997	7, 11	1997 to 1999: Y2K remediation of the core
1998	XML 1.0 a W3C Recommendatio n	10 February 1998	7	
1999	Mars Climate Orbiter lost	23 September 1999; board report 10 November 1999	7, 11	

The 2000s

Year	Landmark	Dates	Ch	The insurer adopts
2000	SOAP 1.1 W3C Note	8 May 2000	8	
2000	Brewer's conjecture	PODC keynote, 19 July 2000	8	

Year	Landmark	Dates	Ch	The insurer adopts
2000	Fielding's dissertation defines REST	2000	8	
2001	Agile Manifesto	11 to 13 February 2001	8	
2001	WSDL 1.1 W3C Note	15 March 2001	8	2007: the bus, with WSDL service contracts
2002	Gilbert and Lynch prove the CAP conjecture	June 2002	8	
2002	Sarbanes-Oxley Act	30 July 2002	8	
2003	Northeast blackout	14 August 2003; task force report April 2004	8	2003: web portal
2003	Evans, Domain-Driven Design	Published 20 August 2003; imprint 2004	8	
2003	Hohpe and Woolf, Enterprise Integration Patterns	Published 10 October 2003; imprint 2004	8	
2004	Fowler, "Strangler Application" (the strangler fig)	29 June 2004; renamed "Strangler Fig Application" 29 April 2019; revised 22 August 2024	8, 12	2004: mainframe operations move to an outsourcer
2004	Amazon Simple Queue Service beta	Beta 3 November 2004; production July 2006	8	

Year	Landmark	Dates	Ch	The insurer adopts
2006	Amazon S3 and EC2	S3 14 March 2006; EC2 beta August 2006	8	2006 to 2009: core rewrite, abandoned
2007	Nygard, Release It!	March 2007	8	
2007	W3C drafts on versioning: TAG draft finding; XML Schema 1.1 versioning guide	Working Draft 20 July 2007; draft finding 13 November 2007	11	
2007	Dynamo paper	SOSP, October 2007	8	
2008	Google App Engine preview; Windows Azure announced	7 April 2008; 27 October 2008 (Azure available 1 February 2010)	8	
2009	"10+ Deploys Per Day"; first devopsdays	23 June 2009; 30 and 31 October 2009	8	

The 2010s

Year	Landmark	Dates	Ch	The insurer adopts
2010	Humble and Farley, Continuous Delivery	27 July 2010	9	
2011	Netflix's Simian Army and Chaos Monkey	Post 19 July 2011; code 30 July 2012	9	
2011	Kafka presented at the NetDB workshop	June 2011	9	

Year	Landmark	Dates	Ch	The insurer adopts
2011	Nygard, architecture decision records	15 November 2011	1	
2012	Knight Capital loss	1 August 2012; SEC order 16 October 2013	9	
2013	Docker released	Version 0.1.0 on 23 March 2013	9	2013: broker API behind an API gateway
2013	Kreps, "The Log"	16 December 2013	9	2018: event log and CDC from DB2 only
2014	Lewis and Fowler, "Microservices"	25 March 2014	9	2016: services and events at the edge
2014	Kelly Review of the Co-operative Bank	30 April 2014	8	
2014	Kubernetes announced	10 June 2014; 1.0 on 21 July 2015	9	
2014	AWS Lambda preview	13 November 2014	9	
2015	Borg paper	EuroSys, April 2015	9	
2016	Site Reliability Engineering	Foreword dated March 2016	9	2019: reserve estimate service and a service mesh
2016	GDPR enters into force	24 May 2016; applies from 25 May 2018	9	

Year	Landmark	Dates	Ch	The insurer adopts
2016	GAO report on federal legacy systems (GAO-16-468)	25 May 2016	12	
2018	Forsgren, Humble and Kim, Accelerate	27 March 2018	2, 9	

The 2020s

Year	Landmark	Dates	Ch	The insurer adopts
2020	CISA directive on SolarWinds Orion	13 December 2020	10	
2021	GitHub Copilot preview	29 June 2021; generally available 21 June 2022	10	
2021	Log4Shell (CVE-2021-44228)	Published 10 December 2021	10, 11	
2022	ChatGPT introduced	30 November 2022	10	
2022	DORA, Regulation (EU) 2022/2554	14 December 2022; applies from 17 January 2025	10	
2023	NIST AI Risk Management Framework 1.0	26 January 2023	10	
2023	Prime Video monitoring service post	22 March 2023	10	2024: modular monolith

Year	Landmark	Dates	Ch	The insurer adopts
2023	NAIC model bulletin on insurers' use of AI	4 December 2023	10	
2024	Moffatt v. Air Canada, 2024 BCCRT 149	14 February 2024	10	
2024	xz backdoor found (CVE-2024-3094)	29 March 2024	10	
2024	CrowdStrike channel file update	19 July 2024; reviews 24 July and 6 August 2024	10	
2024	EU AI Act, Regulation (EU) 2024/1689	In force 1 August 2024; obligations phased in from 2 February 2025	10	
2024	Directive (EU) 2024/2853 on liability for defective products	Signed 23 October 2024; applies to products placed on the market after 9 December 2026	Epilogue	
2024	Model Context Protocol published	25 November 2024	10	2025: agents triage under contracts

Year	Landmark	Dates	Ch	The insurer adopts
2026	Regulation (EU) 2026/1744, the Digital Omnibus on AI	Adopted 8 July 2026, published 24 July 2026; Annex III high-risk rules apply from 2 December 2027, Annex I from 2 August 2028	10	
2026	IEA, Key Questions on Energy and AI	2026	Epilogue	

Glossary

The terms the chapters define. The first chapter in brackets is where each is introduced; entries about the illustrative insurer say so.

Term	Meaning
Abstract data type	A type defined by the operations allowed on it, not by its layout; the representation stays hidden (Liskov and Zilles, 1974) [5]
Accretion	Growth by adding layers around what already runs, rather than replacing it; the insurer's claims system grew this way [Prologue, 2]
Adoption year	The year the insurer started to use something, not its release year; every year in the claims story is one [Brief]
Agent (AI agent)	Software around a language model that plans steps and calls tools; the insurer's agents triage claims and only propose [10]
API gateway	A component at the edge that receives calls to published interfaces and applies authentication, limits and routing before passing them on [9]
Architecture decision record (ADR)	A short document recording one architectural decision with its context, status and consequences (Nygard, 2011) [1]
At-least-once delivery	Accepted messages are delivered one or more times under the service's documented retention and failure conditions; consumers must handle duplicates [12]
Batch	Work collected and processed together in one scheduled run, rather than item by item as it arrives [3]
Binding constraint	The thing that ran out first in a decade and so shaped its designs: machine time, project complexity, the cost of change [Brief]

Term	Meaning
Bohrbug and Heisenbug	A Bohrbug fails again on retry; a Heisenbug is transient, so a backup can mask it (Gray, 1985) [11]
Booked reserves	Recorded liabilities for unpaid claims, including case reserves and incurred-but-not-reported claims; in this illustration, the core updates case reserves nightly and recalculates IBNR monthly [5]
Boundary	A line in a system that data, a call, a change or a request must cross; where an agreement is needed [1]
Bounded context	A part of a large system within which one business vocabulary holds its meaning (Evans, Domain-Driven Design) [8]
Change data capture (CDC)	Reading a database's committed changes and publishing each as an event; the insurer captures DB2 only, not IMS or VSAM [9]
Check digit	An extra digit computed from the others so that a mistyped number can be detected (Luhn's patent, granted 1960) [3]
CICS	IBM's mainframe transaction system for online work; the insurer's claim inquiry, entry and policy lookup run on it [4, 6]
Circuit breaker	A wrapper around a remote call that stops calling a failing service for a while instead of waiting on it (Nygard, Release It!) [8]
Claim of record	The insurer's official version of a claim; in the thread it is created only in the nightly batch [5]
Client-server	A design in which a program on a PC (the client) talks over a network to a program that owns the data (the server) [7]
Closed subroutine	A routine stored once, which a program jumps to and which returns to where it was called; credited to David Wheeler on EDSAC [3]

Term	Meaning
Cohesion	How well a module's parts belong together; Stevens, Myers and Constantine called it "binding" [5]
Contract	The agreed, checkable description of what crosses a boundary: a record layout, a transaction code, an interface description, an event schema, a tool's rules [1]
Control total	A count or sum computed before and after a step and compared, so that lost or changed records show up [3]
Copybook	A file of COBOL record definitions copied into every program that uses the record when it is compiled [5]
CORBA and IDL	The Common Object Request Broker Architecture, a standard for calling objects across a network; its Interface Definition Language (IDL) describes an object's methods neutrally [7]
Coupling	"The measure of the strength of association established by a connection from one module to another" (Stevens, Myers and Constantine, 1974) [5]
Decision Card	The book's one-page record of an architectural decision, in ten fields, from the decision to its change guide [Brief, Appendix A]
Door	The book's word for a point where something crosses a boundary; a filled marker identifies a contract; a hollow red marker flags a missing or unenforced agreement or a missing feed, distinguished by its label [1]
Emulation	Running programs written for one machine on another, which imitates the first; the insurer ran its 1401 programs this way from 1967 [4]

Term	Meaning
End-to-end argument	The principle that some functions can be done correctly only at the ends of a communication, so lower layers should provide them, if at all, only as a performance enhancement (Saltzer, Reed and Clark, 1984) [6]
Enterprise service bus (ESB)	A middleware layer through which services and channels call each other, with routing, transformation of message formats and central logging [8]
Error budget	The amount of unreliability a service may have within its target, used to decide when to slow change and fix reliability [9]
Event log	An ordered, durable record of events that consumers read at their own pace and can replay [9]
Evidence trail	The record kept for each decision a system makes: inputs, the proposal, the check applied and the outcome [1, 10]
Fail-fast	A module that either does the right thing or stops (Gray, 1985); as a process pair it masks Heisenbugs but not a Bohrbug, which fails in the backup too, as at Ariane 501 [6, 11]
Fallacies of distributed computing	A list of false assumptions about networks, such as "the network is reliable", circulated at Sun; L. Peter Deutsch says he added four to four already known there; later accounts credit Bill Joy and James Gosling [7]
Hash total	A sum of identifying numbers, not a business quantity; it detects many errors but misses permutations and compensating changes [3]
Idempotency	An operation whose effect is the same whether applied once or several times [12]
IMS and DL/I	IBM's hierarchical database and transaction system, and its data language; the insurer keeps its open-claim masters in IMS [4, 5]

Term	Meaning
Information hiding	Putting each decision likely to change behind an interface that does not change (Parnas, 1972) [Prologue, 5]
Initial reserve estimate	The first estimate of a claim's cost, made at intake; distinct from booked reserves, which the core still calculates [9]
JCL	Job control language: the statements that tell a mainframe which programs to run in a batch, in what order, with which files [5]
Least privilege	Every program and user should have only the access the job needs (Saltzer and Schroeder, 1975) [5]
Microservices	An application built "as a suite of small services, each running in its own process and communicating with lightweight mechanisms" (Lewis and Fowler, 2014) [9]
Modular monolith	One deployable unit with module lines enforced inside it; the insurer merged intake and the reserve estimate into one in 2024 [10]
Monday rule	The one sentence at the end of each chapter that a technology leader can act on in the next week [Brief]
Policy check	The automated check that accepts or refuses each action an agent proposes, against the tool's contract; not insurance policy data [1, 10]
Release year	The year a vendor or standards body made something available; set against the insurer's adoption year [Brief]
Remote procedure call (RPC)	A call to a procedure on another machine written with the same syntax as a local call (Birrell and Nelson, 1984) [6]
Separation of privilege	Requiring two keys, or two conditions, before a protected action proceeds (Saltzer and Schroeder, 1975); not the same as separation of duties [5]

Term	Meaning
Service mesh	A layer between services that handles their network traffic, security and retries outside the service code [9]
Service-oriented architecture (SOA)	Building enterprise systems as services with published descriptions, called by other systems over a network [8]
Silent failure	A failure in which the system goes on running and raises no error at the boundary where the fault crossed; people learn of it from its consequences, not from the system [Brief, 3]
Strangler fig	Replacing a system gradually by building the new one around its edges until the old one can be removed (Fowler, 2004) [8, 12]
Superseded	A Decision Card status: no longer the target design. It does not mean switched off [Appendix A]
Thread	The book's running example: the illustrative insurer's claims system, followed decade by decade in the insurer's adoption years [Brief]
Three-tier	A design with presentation on the client, business logic on an application server, and data in a database or legacy system [7]
Time-sharing	Sharing one computer among many users at terminals at once, each served in turn so quickly that it seems their own [4]
Triage	Sorting incoming claims by route and urgency; in the thread, agents propose the route and people and contracts decide [1, 10]
Unit record	The punched-card method in which one card holds one record and machines sort, count and tabulate the cards [3]
VSAM	Virtual Storage Access Method, IBM's file access method for keyed and sequential records on its mainframes [5]

Term	Meaning
Waterfall	The later name for the sequence of steps from requirements to operations drawn in Royce's 1970 paper [5]
WSDL	Web Services Description Language: "an XML format for describing network services as a set of endpoints operating on messages" (W3C Note, 2001) [8]
Wrap	Keeping an old layer running unchanged under a new one, as emulation carried the 1401 programs, or behind a new interface [2, 12]

Platform Mapping, as of November 2026

Chapter 12 described the insurer's claims system by its capabilities and their semantics. This appendix maps the capabilities that clouds offer as services, with some related ones the newer layers use, to the products that offer them on three public clouds and on Kubernetes, as of November 2026. It is the only place in the book where current cloud products are mapped, because their names change faster than the ideas; the products named elsewhere in the book are history.

How to read the map

- **Semantics first.** Two products in the same row can differ on what matters most: ordering, delivery guarantee, retention, how they scale and what they cost to run. Read the semantics column, then each product's documentation for those points.
- **The Kubernetes column is the portable one.** It lists Kubernetes APIs and projects of the Cloud Native Computing Foundation (CNCF), with their CNCF maturity level (sandbox, incubating or graduated) on the date checked; where a row also needs a component outside the CNCF, such as a change data capture connector, the cell says so. A portable column is an exit path, not a free one: someone must run it.
- **Open standards carry across columns.** OpenAPI and AsyncAPI describe interfaces; CloudEvents describes an event envelope; OpenTelemetry carries traces and metrics; OCI defines container images. A design that keeps its contracts in these forms can change column with less rework.
- **No prices.** Cost is a shape (per request, per hour, per gigabyte retained), and prices change too often to print.
- **Changes in the past year.** AWS discontinued support for App Mesh on 30 September 2026, so it is not listed. Two agent runtimes were renamed: Azure AI Foundry became Microsoft Foundry in November 2025, and Google Cloud's Agent Engine became Agent Runtime in the Gemini Enterprise Agent Platform in April 2026. NATS (2 November 2026) and OpenTelemetry (11 May 2026) moved to CNCF graduated, as their CNCF project pages record. Azure Data Factory's change data capture resource was in preview on the date checked.

FIGURE D.1

Platform mapping as of November 2026: the capabilities clouds offer as services, by provider

As of November 2026. Product names change: check each vendor's documentation. No prices; cost shapes only.

Capability	Semantics to compare	Azure	AWS	Google Cloud	Kubernetes (portable)
Durable queue	At-least-once delivery; acknowledgement per message; dead-letter path	Azure Service Bus (queues)	Amazon SQS	Cloud Tasks; Pub/Sub	NATS JetStream (CNCF graduated)
Ordered, replayable log	Order per partition key; replay from an offset; retention by time or size	Azure Event Hubs	Amazon Kinesis Data Streams; Amazon MSK	Google Cloud Managed Service for Apache Kafka	Strimzi (CNCF incubating)
Event routing	Route by event type or content; push to subscribers; retry policy	Azure Event Grid	Amazon EventBridge	Eventarc	Knative Eventing (CNCF graduated)
Durable workflow	State kept between steps; retries and timeouts per step; history of each run	Durable Functions; Azure Logic Apps	AWS Step Functions	Workflows	Argo Workflows (CNCF graduated)
Function on demand	Runs per event; scales to zero; limits on run time	Azure Functions	AWS Lambda	Cloud Run functions	Knative Serving (CNCF graduated)
Container platform	OCI images; scale by replicas; rollout and rollback per deployable	Azure Kubernetes Service (AKS)	Amazon EKS	Google Kubernetes Engine (GKE)	Deployment API
API gateway	Authentication, rate limits and routing by version at the edge	Azure API Management	Amazon API Gateway	Apigee; API Gateway	Gateway API
Service-to-service traffic	Mutual TLS, retries and timeouts outside the code; policy per route	Istio-based service mesh add-on for AKS	Amazon VPC Lattice; Amazon ECS Service Connect	Cloud Service Mesh	Istio; Linkerd (both CNCF graduated)
Change data capture	Committed changes in commit order; covers only the stores it can read	Azure Data Factory change data capture (preview)	AWS Database Migration Service (ongoing replication)	Datastream	Strimzi KafkaConnect and KafkaConnector resources (the connector itself is not a CNCF project)
Event schema registry	Versioned schemas; compatibility checks before a producer changes	Schema Registry in Azure Event Hubs	AWS Glue Schema Registry	Pub/Sub schemas	Apicurio Registry (CNCF sandbox)
Tracing and metrics	One trace across hops; sampling; OpenTelemetry (OTLP) intake	Azure Monitor Application Insights	Amazon CloudWatch; AWS X-Ray	Cloud Trace (Google Cloud Observability)	OpenTelemetry Collector, Jaeger, Prometheus (CNCF graduated)

Capability	Semantics to compare	Azure	AWS	Google Cloud	Kubernetes (portable)
Secrets	Central store; rotation; access audited per read	Azure Key Vault	AWS Secrets Manager	Secret Manager	Secret API; External Secrets Operator (CNCF sandbox)
Application policy decision	Allow or deny each action against policy held outside the code	No managed equivalent found (Azure Policy governs cloud resources only)	Amazon Verified Permissions	No managed equivalent found (IAM governs cloud resources; load balancer authorization policies can call an external decision point)	Open Policy Agent (CNCF graduated)
Agent runtime with tool calls	Hosts agents; tool calls through declared interfaces; identity and logs per call	Microsoft Foundry Agent Service	Amazon Bedrock AgentCore	Agent Runtime (Gemini Enterprise Agent Platform)	kagent (CNCF sandbox)

A dated map, not a recommendation: compare the semantics column first, and recheck every name against the vendor's documentation before you use it.

Vendor and project documentation (Microsoft Learn, AWS documentation, Google Cloud documentation, kubernetes.io, cncf.io), every row checked on 20 November 2026; the pages are listed in Appendix E.

The MQ payment queue belongs in the durable-queue row. IBM describes persistent MQ messages as assured of once and once only delivery; the insurer's payment path is still designed for at-least-once delivery, because a message can arrive twice when it is redelivered after a failure upstream or read by a consumer outside a syncpoint, so the consumer discards duplicates. The map has no row for the rest of the core: the nightly batch and its scheduling, IMS and CICS. Running them on a cloud means a vendor's runtime or a rewrite, and that is a migration, not a product choice. AWS offered a managed runtime for such workloads in AWS Mainframe Modernization, but as of November 2026 its documentation says that the managed runtime has stopped accepting new customers and that the self-managed experience is no longer open to new customers. Nor has the map a row for central routing and transformation, the 2007 bus: it is being retired route by route (Chapter 12), and its routes move to rows that are here, such as event routing, durable workflow and the API gateway.

Where a cell reads "no managed equivalent found", the capability is still needed; it is built from the Kubernetes column or from a vendor's general tools. The map is refreshed once a year. Every row was checked on 20 November 2026, and Appendix E lists the page that each name was checked against.

APPENDIX E

Sources

Every dated fact in this book was checked against the source listed for its chapter in November 2026. Primary sources come first: original papers, standards, statutes, judgments, official reports, vendor announcements and institutional histories. This list holds 368 chapter source entries, including repeated works and notes on the illustrative case; the claim-by-claim record behind it is kept with the book and is available on request.

Reader's Brief: The Binding Constraint and Six Questions

- Eliyahu M. Goldratt and Jeff Cox, The Goal, North River Press, first published 1984 (publisher page) https://northriverpress.com/the-goal-30th-anniversary-edition/; North River Press, The Five Focusing Steps ("The Theory of Constraints was first introduced in Eli Goldratt's classic business novel The Goal") https://northriverpress.com/wp-content/uploads/2018/01/Free-download-5FS.pdf
- Bertrand Meyer, publication list (ETH Zurich) https://se.inf.ethz.ch/~meyer/publications/; IEEE Computer, October 1992 contents https://www.computer.org/csdl/magazine/co/1992/10
- ISO/IEC 25010:2011 https://www.iso.org/standard/35733.html; ISO/IEC 25010:2023 https://www.iso.org/standard/78176.html
- J. Gray, Tandem TR 85.7 (Carnegie Mellon course copy) https://www.andrew.cmu.edu/course/15-440/assets/READINGS/gray1985.pdf
- J. H. Saltzer and M. D. Schroeder, paper text (University of Virginia copy) https://www.cs.virginia.edu/~evans/cs551/saltzer/
- IBM Redbooks, CICS Transaction Server from Start to Finish, SG24-7952, section 1.9.1 https://www.redbooks.ibm.com/redbooks/pdfs/sg247952.pdf
- IBM, Celebrating 40 years of Db2: Running the world's mission critical workloads (the page title says 40 years; the address keeps an older slug) https://www.ibm.com/new/announcements/celebrating-30-years-of-db2-running-the-worlds-mission-critical-workloads
- TOCICO, The Foundation of the Theory of Constraints ("The five focusing steps (5FS) process originated in 1987") https://learningcenter.tocico.org/courses/the-foundation-of-the-theory-of-constraints-what-is-theory-of-constraints; North River Press, The Five Focusing Steps, adapted from Goldratt, What Is This Thing Called Theory of Constraints (1990) https://northriverpress.com/wp-content/uploads/2018/01/Free-download-5FS.pdf
- ACM record (DOI 10.1145/361598.361623) https://dl.acm.org/doi/10.1145/361598.361623
- The MIT Press, Out of the Crisis (reissue, 2018), publisher record https://api.crossref.org/works/10.7551/mitpress/11457.001.0001; review record, Quality and Reliability Engineering International, 1988 (Out of the Crisis, MIT/CAES, 1986) https://doi.org/10.1002/qre.4680040119
- NIST, Cybersecurity Framework 1.0 https://www.nist.gov/system/files/documents/cyberframework/cybersecurity-framework-021214.pdf
- Google, SRE books https://sre.google/books/; https://sre.google/sre-book/table-of-contents/
- ISO/IEC 25010:2023 https://www.iso.org/standard/78176.html

One Claim, Seventy Years

- IBM, The IBM punched card https://www.ibm.com/history/punched-card
- IBM, General Information Manual, IBM 1401 Data Processing System: From Control Panel to Stored Program, F20-208 (1959) https://bitsavers.org/pdf/ibm/1401/F20-208_General_Information_Manual_IBM_1401_From_Control_Panel_to_Stored_Program_1959.pdf
- ACM record (DOI 10.1145/361598.361623) https://dl.acm.org/doi/10.1145/361598.361623
- J. Friedland, Estimating Unpaid Claims Using Basic Techniques, Casualty Actuarial Society, version 3, July 2010 https://www.casact.org/library/studynotes/Friedland_estimating.pdf

Chapter 1. What Architecture Decides

- D. E. Perry and A. L. Wolf, paper text (University of Victoria copy) https://webhome.csc.uvic.ca/~hausi/480/papers/perry.pdf
- P. Kruchten, paper text (University of British Columbia copy) https://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf
- ISO/IEC/IEEE 42010 site, Defining architecture http://www.iso-architecture.org/ieee-1471/defining-architecture.html; ISO/IEC/IEEE 42010 home page ("10 JUL 2007 ISO has published ISO/IEC 42010:2007") http://www.iso-architecture.org/ieee-1471/
- W. Kunz and H. W. J. Rittel, scanned paper (eScholarship) https://escholarship.org/uc/item/5cj786v8
- M. Nygard, Cognitect blog https://cognitect.com/blog/2011/11/15/documenting-architecture-decisions
- ACM record (DOI 10.1145/58566.59297) https://dl.acm.org/doi/10.1145/58566.59297
- Thoughtworks, Technology Radar, Lightweight Architecture Decision Records https://www.thoughtworks.com/radar/techniques/lightweight-architecture-decision-records
- Crossref record (DOI 10.1109/MS.2005.27) https://api.crossref.org/works/10.1109/MS.2005.27
- Crossref record (DOI 10.1109/WICSA.2005.61) https://api.crossref.org/works/10.1109/WICSA.2005.61
- NASA, Mars Climate Orbiter Mishap Investigation Board Phase I Report, 10 November 1999 https://llis.nasa.gov/llis_lib/pdf/1009464main1_0641-mr.pdf
- ISO, ISO/IEC/IEEE 42010:2022 Software, systems and enterprise: Architecture description (catalogue page) https://www.iso.org/standard/74393.html

Chapter 2. The Binding Constraint, and How to Read Old Designs

- Computer History Museum, Software Becomes a Product https://www.computerhistory.org/revolution/mainframe-computers/7/172
- IT Revolution, Accelerate https://itrevolution.com/product/accelerate/
- Eliyahu M. Goldratt and Jeff Cox, The Goal, North River Press, first published 1984 (publisher page) https://northriverpress.com/the-goal-30th-anniversary-edition/
- IBM, Information Management Systems https://www.ibm.com/history/information-management-system
- IBM, Celebrating 40 years of Db2: Running the world's mission critical workloads (the page title says 40 years; the address keeps an older slug) https://www.ibm.com/new/announcements/celebrating-30-years-of-db2-running-the-worlds-mission-critical-workloads
- IBM Redpaper, WebSphere MQ Primer https://www.redbooks.ibm.com/redpapers/pdfs/redp0021.pdf
- OMG, CORBA History https://www.omg.org/corba/history_of_corba.htm
- CERN, A short history of the Web https://home.cern/science/computing/the-birth-of-the-web/short-history-web/; the proposal https://info.cern.ch/Proposal.html
- Docker, Docker: Nine Years YOUNG https://www.docker.com/blog/docker-nine-years-young/
- D. M. Ritchie, The Evolution of the Unix Time-sharing System (Nokia Bell Labs) https://www.nokia.com/bell-labs/about/dennis-m-ritchie/hist.html; MIT copy https://pdos.csail.mit.edu/6.828/2005/readings/ritchie79evolution.html
- RAND, P-4947 full text https://www.rand.org/content/dam/rand/pubs/papers/2008/P4947.pdf
- DORA, Accelerate State of DevOps Report 2019 https://dora.dev/publications/pdf/state-of-devops-2019.pdf
- DORA, DORA's software delivery performance metrics https://dora.dev/guides/dora-metrics/
- IBM, The IBM 1401 https://www.ibm.com/history/1401
- IBM Redbooks, CICS Transaction Server from Start to Finish, SG24-7952, section 1.9.1 https://www.redbooks.ibm.com/redbooks/pdfs/sg247952.pdf
- IBM System/360 Model 30 Functional Characteristics, GA24-3231 http://www.bitsavers.org/pdf/ibm/360/functional_characteristics/GA24-3231-7_360-30_funcChar.pdf
- ACM record (DOI 10.1145/362384.362685) https://dl.acm.org/doi/10.1145/362384.362685
- ACM record (DOI 10.1145/1460833.1460871) https://dl.acm.org/doi/10.1145/1460833.1460871
- E. Brewer, keynote slides https://people.eecs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf
- D. M. Ritchie, The Evolution of the Unix Time-sharing System (MIT copy) https://pdos.csail.mit.edu/6.828/2005/readings/ritchie79evolution.html
- IBM, System/360 https://www.ibm.com/history/system-360
- Crossref record with abstract (DOI 10.1145/359511.359522) https://api.crossref.org/works/10.1145/359511.359522

- Crossref record (DOI 10.1109/MC.2012.37) https://api.crossref.org/works/10.1109/MC.2012.37; text republished by InfoQ https://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed/
- IBM, The relational database https://www.ibm.com/history/relational-database
- E. Brewer, CAP Twelve Years Later: How the "Rules" Have Changed, IEEE Computer 45(2), 2012, text republished by InfoQ https://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed/

Chapter 3. Before 1960: Wiring the Machine

- Google Patents, US2950048A https://patents.google.com/patent/US2950048A/en
- US Census Bureau, Tabulation and Processing https://www.census.gov/about/history/bureau-history/census-innovations/technology/tabulation-and-processing.html
- IBM, Accounting Machine Type 407: Preliminary Manual of Information, form 22-5765-0, 1949, Computer History Museum catalog 102724936 https://www.computerhistory.org/collections/catalog/102724936
- Computer History Museum, catalog 102724934, IBM 407 test control panel manual https://www.computerhistory.org/collections/catalog/102724934
- Illustrative thread; no external source.
- Computer History Museum, catalog 102724622 https://www.computerhistory.org/collections/catalog/102724622
- University of Cambridge Computer Laboratory, EDSAC 1 and after https://www.cl.cam.ac.uk/events/EDSAC99/history.html
- IEEE Computer Society, Computer Pioneers: David J. Wheeler https://history.computer.org/pioneers/wheeler.html
- ACM Digital Library, doi 10.1145/609784.609816 https://dl.acm.org/doi/10.1145/609784.609816
- LEO Computers Society https://www.leo-computers.org.uk/?p=3879
- Centre for Computing History, 30th November 1951 https://computinghistory.org.uk/det/66463/30th-November-1951-a-momentous-day; LEO Computers Society https://leo-computers.org.uk/
- M. E. Davis, The Use of Electronic Data Processing Systems in the Life Insurance Business, Proceedings of the Eastern Joint Computer Conference, December 1953 https://bitsavers.trailing-edge.com/pdf/afips/1953-12__%2304.pdf
- MetLife, Our History https://prv.metlife.com/global/company/our-history/
- IBM, The IBM 650 https://www.ibm.com/history/650
- Backus et al., The FORTRAN automatic coding system, doi 10.1145/1455567.1455599 https://dl.acm.org/doi/10.1145/1455567.1455599
- J. Backus, The History of FORTRAN I, II, and III, ACM HOPL 1978 https://users-cs.au.dk/askarov/dovs/p165-backus.pdf
- CODASYL COBOL Journal of Development 1968, NBS Handbook 106 https://nvlpubs.nist.gov/nistpubs/Legacy/hb/nbshandbook106.pdf
- IBM Type 405 Principles of Operation, Computer History Museum archive https://archive.computerhistory.org/resources/access/text/2022/11/102698809-05-01-acc.pdf
- IBM 705 Generalized Merge Program, form 32-7626, 1957 https://archive.org/details/bitsavers_ibm7053276gePgm1957_5277626
- Computer History Museum, catalog 102623295 https://www.computerhistory.org/collections/catalog/102623295

Chapter 4. The 1960s: The Terminal and the Crisis

- NASA, NSSDCA Master Catalog, Mariner 1 https://nssdc.gsfc.nasa.gov/nmc/spacecraft/display.action?id=MARIN1
- NASA History, 60 Years Ago: Mariner 1 Launch Attempt to Venus https://www.nasa.gov/history/60-years-ago-mariner-1-launch-attempt-to-venus/
- IBM, The IBM 1401 https://www.ibm.com/history/1401
- Computer History Museum 1401 restoration team, 1401 Origins https://ibm1401.computerhistory.org/1401Origins.html
- The Computer Museum, 1401 batch processing exhibit https://tcm.computerhistory.org/exhibits/1401BatchProcessing.pdf
- J. H. Saltzer, tribute to F. J. Corbató, MIT https://web.mit.edu/saltzer/www/publications/CorbatoTribute.pdf
- ACM Digital Library, doi 10.1145/1460833.1460871 https://dl.acm.org/doi/10.1145/1460833.1460871
- Dartmouth College, BASIC at 50 https://www.dartmouth.edu/basicfifty/
- IBM, Sabre https://www.ibm.com/history/sabre
- Computer History Museum, Timeline of Computer History, 1964 https://www.computerhistory.org/timeline/1964/
- IBM, System/360 https://www.ibm.com/history/system-360
- Amdahl, Blaauw and Brooks, Architecture of the IBM System/360, IBM Journal of Research and Development 8(2), April 1964 https://www.ece.ucdavis.edu/~vojin/CLASSES/EEC272/S2005/Papers/IBM360-Amdahl_april64.pdf

- IBM System/360 Model 30 Functional Characteristics, GA24-3231 http://www.bitsavers.org/pdf/ibm/360/functional_characteristics/GA24-3231-7_360-30_funcChar.pdf
- C. Boyer, The 360 Revolution, IBM https://www.ibm.com/support/pages/zvm/history/360rev.pdf
- NATO Software Engineering Conference 1968 report http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF
- NATO Software Engineering Techniques 1969 report http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1969.PDF
- E. W. Dijkstra Archive, EWD215 https://www.cs.utexas.edu/~EWD/transcriptions/EWD02xx/EWD215.html
- M. E. Conway, author's site https://www.melconway.com/Home/Committees_Paper.html
- ACM, Charles W. Bachman oral history https://amturing.acm.org/pdf/BachmanTuringTranscript.pdf
- IBM, Information Management System https://www.ibm.com/history/information-management-system
- IBM, CICS Basics https://public.dhe.ibm.com/software/data/sw-library/CICS_Basics.pdf; IBM Redbook SG24-7952 https://www.redbooks.ibm.com/redbooks/pdfs/sg247952.pdf
- Illustrative thread; no external source.
- IBM, System/360 Model 30 1401 Compatibility Feature, A24-3255-1, April 1964 http://www.bitsavers.org/pdf/ibm/360/1401_emulator/A24-3255-1_Model30_1401_Compatibility_Apr64.pdf
- IBM Documentation, History of IMS: Beginnings at NASA https://www.ibm.com/docs/zosbasics/com.ibm.imsintro.doc.intro/ipoind0011003710.htm
- M. Fowler, Original Strangler Fig Application https://martinfowler.com/bliki/OriginalStranglerFigApplication.html
- NASA Science, Mariner 1 https://science.nasa.gov/mission/mariner-1/

Chapter 5. The 1970s: Structure and Change

- US General Accounting Office, NORAD's Missile Warning System: What Went Wrong?, MASAD-81-30, 15 May 1981 https://www.gao.gov/assets/masad-81-30.pdf
- Illustrative thread; no external source.
- IBM, OS/VS Virtual Storage Access Method (VSAM) Planning Guide http://www.bitsavers.org/pdf/ibm/370/OS_VS2/Release_1_1972/
- IBM, System/370 https://www.ibm.com/history/system-370
- IBM, IMS/VS General Information Manual https://kuiper.mirrorservice.org/sites/www.bitsavers.org/pdf/ibm/370/IMS_VS/GH20-1260-0_IMS_VS_General_Information_Manual_Jan73.pdf; Computer History Museum, catalog 102786536 https://www.computerhistory.org/collections/catalog/102786536
- D. L. Parnas, On the Criteria To Be Used in Decomposing Systems into Modules https://wstomv.win.tue.nl/edu/2ip30/references/criteria_for_modularization.pdf
- ACM Digital Library, doi 10.1145/942572.807045 https://dl.acm.org/doi/10.1145/942572.807045
- IBM Systems Journal, doi 10.1147/sj.132.0115 https://doi.org/10.1147/sj.132.0115
- W. W. Royce, Managing the Development of Large Software Systems https://www.praxisframework.org/files/royce1970.pdf
- F. P. Brooks, curriculum vitae https://www.cs.unc.edu/~brooks/FPB_BIO.CV.06.2006.pdf; text at https://archive.org/details/MythicalManMonth
- L. A. Belady and M. M. Lehman, author copy https://www.cs.mcgill.ca/~martin/teaching/comp762-fall-2005/BL1976a.pdf
- M. M. Lehman, Proceedings of the IEEE, doi 10.1109/PROC.1980.11805 https://doi.org/10.1109/PROC.1980.11805
- ACM Digital Library, doi 10.1145/362384.362685 https://dl.acm.org/doi/10.1145/362384.362685
- Chamberlin et al., A History and Evaluation of System R, CACM 24(10), October 1981 https://dsf.berkeley.edu/cs262/2005/SystemR.pdf
- Datapro report M11-822, Tandem NonStop, January 1979 https://ftp.mirrorservice.org/sites/www.bitsavers.org/pdf/datapro/datapro_reports_70s-90s/Tandem/M11-822-10_7901_Tandem_Non-Stop.pdf
- Springer, doi 10.1007/3-540-08755-9_9 https://doi.org/10.1007/3-540-08755-9_9
- Amdahl, Blaauw and Brooks, Architecture of the IBM System/360, IBM Journal of Research and Development 8(2), April 1964 https://www.ece.ucdavis.edu/~vojin/CLASSES/EEC272/S2005/Papers/IBM360-Amdahl_april64.pdf
- IBM press release, 30 June 1970, IBM Archives (archived copy) http://web.archive.org/web/2006/http://www.ibm.com/ibm/history/exhibits/mainframe/mainframe_PR370.html
- In re Equity Funding Corp. of America Securities Litigation, 375 F. Supp. 1378 (J.P.M.L. 1974) https://www.govinfo.gov/content/pkg/USCOURTS-jpml-1_73-F-00142/pdf/USCOURTS-jpml-1_73-F-00142-0.pdf
- J. H. Saltzer and M. D. Schroeder, The Protection of Information in Computer Systems https://web.mit.edu/Saltzer/www/publications/protection/Basic.html
- RAND, R-609-1 https://www.rand.org/pubs/reports/R609-1.html

- IBM Documentation, History of IMS: Beginnings at NASA https://www.ibm.com/docs/zosbasics/com.ibm.imsintro.doc.intro/ipoind0011003710.htm
- IBM, Enterprise COBOL for z/OS, COPY statement https://www.ibm.com/docs/en/cobol-zos/6.4.0?topic=statements-copy-statement

Chapter 6. The 1980s: Computing Leaves the Glass Room

- IBM, "The IBM PC" https://www.ibm.com/history/personal-computer
- IBM Redbooks, CICS Transaction Server from Start to Finish, SG24-7952 https://www.redbooks.ibm.com/redbooks/pdfs/sg247952.pdf; IBM Support, CICS history ("Designed to support 50 BTAM terminals") https://www.ibm.com/support/pages/node/658565
- Computer History Museum, catalog 102704058 https://www.computerhistory.org/collections/catalog/102704058
- Computer History Museum, Revolution, "Software: Putting PCs to Work" https://www.computerhistory.org/revolution/personal-computers/17/305
- Computer History Museum, catalog 102651720 https://www.computerhistory.org/collections/catalog/102651720
- RFC 801, NCP/TCP Transition Plan https://www.rfc-editor.org/rfc/rfc801.txt
- CACM https://doi.org/10.1145/360248.360253; copy https://www.cl.cam.ac.uk/teaching/0708/DigiCommI/metcalfe1976ethernet.pdf
- IEEE SA, 802.3-1985 https://standards.ieee.org/standard/802_3-1985.html
- ACM TOCS https://doi.org/10.1145/2080.357392; copy read https://web.eecs.umich.edu/~mosharaf/Readings/RPC.pdf
- ACM TOCS https://doi.org/10.1145/357401.357402; author copy with the 1981 note https://web.mit.edu/Saltzer/www/publications/endtoend/endtoend.pdf
- IBM, "The relational database" https://www.ibm.com/history/relational-database
- IBM, "Celebrating 40 years of Db2" https://www.ibm.com/new/announcements/celebrating-30-years-of-db2-running-the-worlds-mission-critical-workloads
- Report https://jimgray.azurewebsites.net/papers/TandemTR85.7_WhyDoComputersStop.pdf
- Paper (reproduction) https://worrydream.com/refs/Brooks_1986_-_No_Silver_Bullet.pdf; DOI record https://doi.org/10.1109/MC.1987.1663532
- R. P. Gabriel, "Worse Is Better" https://www.dreamsongs.com/WorseIsBetter.html
- R. P. Gabriel, "The Rise of Worse is Better" https://www.dreamsongs.com/RiseOfWorseIsBetter.html
- Leveson and Turner, "An Investigation of the Therac-25 Accidents", IEEE Computer 26(7), July 1993, pp. 18 to 41 https://doi.org/10.1109/MC.1993.274940; copy read https://www.se.rit.edu/~swen-342/resources/Leveson,%20Turner%20-%20An%20Investigation%20of%20the%20Therac-25%20Accidents%20-%20IEEE%20Computer%20v26n7%20-%201993-07.pdf
- James Martin, Application Development Without Programmers, Prentice-Hall, 1982 https://archive.org/details/applicationdevel00mart
- GAO, Computer Security: Virus Highlights Need for Improved Internet Management, GAO/IMTEC-89-57, June 1989 https://www.gao.gov/assets/imtec-89-57.pdf
- The series thread for the invented insurer (illustrative, not a historical claim)

Chapter 7. The 1990s: Integration and the Web

- CERN, "30 years of a free and open Web" https://home.cern/30-years-free-and-open-web/
- OMG, "History of CORBA" https://www.omg.org/corba/history_of_corba.htm; OMG specification catalogue https://www.omg.org/spec/CORBA/
- OMG, CORBA 2.0 specification https://www.omg.org/spec/CORBA/2.0/PDF
- Sun Microsystems Laboratories report (archived original URL) https://web.archive.org/web/2010id_/http://research.sun.com/techrep/1994/smli_tr-94-29.pdf
- Sun, java.sun.com/people/jag/Fallacies.html (archived 8 October 1999) https://web.archive.org/web/19991008110904/http://java.sun.com/people/jag/Fallacies.html
- Software Engineering Radio, episode 470, "L Peter Deutsch on the Fallacies of Distributed Computing", 2021 https://se-radio.net/2021/07/episode-470-l-peter-deutsch-on-the-fallacies-of-distributed-computing/
- InformIT (Pearson) https://www.informit.com/store/design-patterns-elements-of-reusable-object-oriented-9780201633610
- Author's reprint https://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf; DOI record https://doi.org/10.1109/52.469759

- IBM Redbooks, "Universal Messaging with IBM WebSphere MQ", TIPS0925, 2012 https://www.redbooks.ibm.com/abstracts/tips0925.html
- W3C, "Information Management: A Proposal" https://www.w3.org/History/1989/proposal.html
- RFC 3875, The Common Gateway Interface (CGI) Version 1.1, October 2004 https://www.rfc-editor.org/rfc/rfc3875.txt
- RFC 1945 https://www.rfc-editor.org/rfc/rfc1945.txt; RFC 2068 https://www.rfc-editor.org/rfc/rfc2068.txt
- Sun press release, "Sun Brings True Interactivity to the World Wide Web", archived copy http://web.archive.org/web/20051207025921/http://www.sun.com:80/smi/Press/sunflash/1995-05/sunflash.950523.3421.html
- Sun press release "JavaSoft Ships Java 1.0", archived copy http://web.archive.org/web/20031014103813/http://java.sun.com:80/pr/1996/jan/pr960123-01.html
- W3C https://www.w3.org/TR/1998/REC-xml-19980210
- NASA, Mars Climate Orbiter Mishap Investigation Board Phase I Report, 10 November 1999 https://llis.nasa.gov/llis_lib/pdf/1009464main1_0641-mr.pdf
- GAO, Year 2000 Computing Crisis: An Assessment Guide, GAO/AIMD-10.1.14, September 1997 https://www.gao.gov/assets/aimd-10.1.14.pdf
- IBM, The Year 2000 and 2-Digit Dates: A Guide for Planning and Implementation, GC28-1251-08, February 1998 https://publibfi.dhe.ibm.com/epubs/pdf/da61a106.pdf
- M. Henning, "The Rise and Fall of CORBA", ACM Queue 4(5), June 2006 https://queue.acm.org/detail.cfm?id=1142044
- W3C archive https://www.w3.org/People/Berners-Lee/1991/08/art-6484.txt
- The series thread for the invented insurer (illustrative)
- J. Cardena, From Tally to Token https://blog.jcardena.com/en/ebooks/from-tally-to-token/

Chapter 8. The 2000s: Services and the Team

- US Government Publishing Office, Public Law 107-204 https://www.govinfo.gov/content/pkg/PLAW-107publ204/html/PLAW-107publ204.htm
- W3C https://www.w3.org/TR/2000/NOTE-SOAP-20000508/
- W3C https://www.w3.org/TR/2001/NOTE-wsdl-20010315
- ACORD, 50th Anniversary Member Report, 2020 https://www.acord.org/docs/default-source/membership-programs-info/annual-member-reports/memberreport_2020_50thanniversary_digital.pdf
- Federal Register, Health Insurance Reform: Standards for Electronic Transactions https://www.govinfo.gov/content/pkg/FR-2000-08-17/pdf/00-20820.pdf
- R. T. Fielding, Architectural Styles and the Design of Network-based Software Architectures https://ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
- Sir Christopher Kelly, Failings in management and governance (Kelly Review), The Co-operative Group https://www.co-operative.coop/investors/kelly-review; full text copy https://saveourbank.coop/docs/kelly-review.pdf
- Manifesto for Agile Software Development https://agilemanifesto.org/; History https://agilemanifesto.org/history.html
- Pearson https://www.pearson.com/en-gb/subject-catalog/p/domain-driven-design-tackling-complexity-in-the-heart-of-software/P200000009375/9780321125217
- Pearson https://www.pearson.com/store/en-us/p/enterprise-integration-patterns-designing-building-and-deploying-messaging-solutions/P200000009198/9780321200686
- Pragmatic Bookshelf (archived 2007) https://web.archive.org/web/2007/http://www.pragprog.com/titles/mnee; table of contents http://media.pragprog.com/titles/mnee/toc.pdf
- M. Fowler, "CircuitBreaker" https://martinfowler.com/bliki/CircuitBreaker.html
- O'Reilly Velocity 2009 session page (archived) https://web.archive.org/web/2009/http://en.oreilly.com/velocity2009/public/schedule/detail/7641
- devopsdays (archived October 2009) https://web.archive.org/web/20091026023812/http://www.devopsdays.org:80/
- AWS What's New https://aws.amazon.com/about-aws/whats-new/2004/11/03/introducing-the-amazon-simple-queue-service/
- AWS News Blog, "Amazon Simple Queue Service Released", 13 July 2006 https://aws.amazon.com/blogs/aws/amazon_simple_q/
- Amazon press release https://press.aboutamazon.com/2006/3/amazon-web-services-launches
- AWS What's New https://aws.amazon.com/about-aws/whats-new/2006/08/24/announcing-amazon-elastic-compute-cloud-amazon-ec2---beta/; AWS News Blog https://aws.amazon.com/blogs/aws/amazon_ec2_beta/
- Google App Engine Blog https://googleappengine.blogspot.com/2008/04/introducing-google-app-engine-our-new.html
- Microsoft News https://news.microsoft.com/source/2008/10/27/microsoft-unveils-windows-azure-at-professional-developers-conference/

- Official Microsoft Blog https://blogs.microsoft.com/blog/2010/02/01/windows-azure-general-availability/
- E. Brewer, "Towards Robust Distributed Systems" https://people.eecs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf
- ACM https://doi.org/10.1145/564585.564601; author copy https://www.comp.nus.edu.sg/~gilbert/pubs/BrewersConjecture-SigAct.pdf
- IEEE Computer https://doi.org/10.1109/MC.2012.37; authorised reprint https://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed/
- DeCandia et al., "Dynamo: Amazon's Highly Available Key-value Store" https://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf
- U.S.-Canada Power System Outage Task Force, Final Report on the August 14, 2003 Blackout in the United States and Canada: Causes and Recommendations, April 2004 https://www.energy.gov/sites/default/files/oeprod/DocumentsandMedia/BlackoutFinal-Web.pdf
- NAIC, Model Regulation 205 https://content.naic.org/sites/default/files/model-law-205.pdf
- 45 CFR 160.103 https://www.ecfr.gov/current/title-45/section-160.103; 42 U.S.C. 300gg-91 https://www.govinfo.gov/content/pkg/USCODE-2023-title42/html/USCODE-2023-title42-chap6A-subchapXXV-partC-sec300gg-91.htm
- M. Fowler https://martinfowler.com/bliki/OriginalStranglerFigApplication.html
- The series thread for the invented insurer (illustrative)
- Public Law 107-105 https://www.congress.gov/107/plaws/publ105/PLAW-107publ105.pdf
- O'Reilly catalog https://www.oreilly.com/library/view/enterprise-service-bus/0596006756/
- OASIS https://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf
- Code of Federal Regulations, title 45, part 162, 2002 edition https://www.govinfo.gov/content/pkg/CFR-2002-title45-vol1/pdf/CFR-2002-title45-vol1-part162.pdf

Chapter 9. The 2010s: Many Teams, One Platform

- James Lewis and Martin Fowler, Microservices https://martinfowler.com/articles/microservices.html
- InformIT (Pearson), Continuous Delivery https://www.informit.com/store/continuous-delivery-reliable-software-releases-through-9780321601919; continuousdelivery.com https://continuousdelivery.com/
- The Twelve-Factor App https://12factor.net/; archived capture of 24 November 2011 http://web.archive.org/web/20111124064857/http://www.12factor.net:80/
- IT Revolution, Accelerate https://itrevolution.com/product/accelerate/
- Docker (moby) project change log, CHANGELOG.md at v1.0.0 https://github.com/moby/moby/blob/v1.0.0/CHANGELOG.md
- Google Cloud Platform Blog, An update on container support on Google Cloud Platform, 10 June 2014 https://cloudplatform.googleblog.com/2014/06/an-update-on-container-support-on-google-cloud-platform.html
- Google Cloud Platform Blog, Kubernetes V1 Released, 21 July 2015 https://cloudplatform.googleblog.com/2015/07/Kubernetes-V1-Released.html
- Google Research, publication page https://research.google/pubs/large-scale-cluster-management-at-google-with-borg/
- EuroSys 2015 conference site (archived copy) https://web.archive.org/web/2015/http://eurosys2015.labri.fr/
- AWS News Blog, AWS Lambda: Run Code in the Cloud https://aws.amazon.com/blogs/aws/run-code-cloud/
- LinkedIn Engineering (archived copy) https://web.archive.org/web/2014/https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying
- Netflix Technology Blog https://netflixtechblog.com/the-netflix-simian-army-16e57fbab116
- Netflix Tech Blog, Chaos Monkey Released Into The Wild, 30 July 2012 (archived copy) https://web.archive.org/web/2012/http://techblog.netflix.com/2012/07/chaos-monkey-released-into-wild.html
- Google, Site Reliability Engineering, foreword https://sre.google/sre-book/foreword/; https://sre.google/books/
- EUR-Lex, Regulation (EU) 2016/679 https://eur-lex.europa.eu/eli/reg/2016/679/oj; (text read from the Publications Office copy http://publications.europa.eu/resource/celex/32016R0679)
- SEC, In the Matter of Knight Capital Americas LLC, Release No. 70694, 16 October 2013 https://www.sec.gov/litigation/admin/2013/34-70694.pdf
- SEC Release No. 70694, as above
- The series thread for the invented insurer (illustrative)
- Kreps, Narkhede, Rao, NetDB 2011 paper https://www.microsoft.com/en-us/research/wp-content/uploads/2017/09/Kafka.pdf
- Segment (Twilio) engineering blog https://segment.com/blog/goodbye-microservices/; (now https://www.twilio.com/en-us/blog/developers/best-practices/goodbye-microservices)

- DORA, DORA metrics history https://dora.dev/insights/dora-metrics-history/; DORA's software delivery performance metrics https://dora.dev/guides/dora-metrics/
- Google, Site Reliability Engineering, chapter 3, Embracing Risk https://sre.google/sre-book/embracing-risk/
- Microsoft Visual Studio Blog, Introducing Visual Studio IntelliCode https://devblogs.microsoft.com/visualstudio/introducing-visual-studio-intellicode/
- Apache Kafka, Introduction, Main Concepts and Terminology https://kafka.apache.org/intro

Chapter 10. The 2020s (Our View): When AI Writes the Code

- GitHub Blog, Introducing GitHub Copilot: your AI pair programmer https://github.blog/2021-06-29-introducing-github-copilot-ai-pair-programmer/
- GitHub Blog, GitHub Copilot is generally available to all developers https://github.blog/2022-06-21-github-copilot-is-generally-available-to-all-developers/
- OpenAI, Introducing ChatGPT https://openai.com/index/chatgpt/
- CISA, ED 21-01 https://www.cisa.gov/news-events/directives/ed-21-01-mitigate-solarwinds-orion-code-compromise
- CVE record (CVE Program API) https://cveawg.mitre.org/api/cve/CVE-2021-44228; NIST NVD https://nvd.nist.gov/vuln/detail/CVE-2021-44228; Apache Log4j security page https://logging.apache.org/log4j/2.x/security.html
- Prime Video Tech (archived copy) https://web.archive.org/web/2023/https://www.primevideotech.com/video-streaming/scaling-up-the-prime-video-audio-video-monitoring-service-and-reducing-costs-by-90
- Anthropic, Introducing the Model Context Protocol https://www.anthropic.com/news/model-context-protocol
- Civil Resolution Tribunal, Moffatt v. Air Canada, 2024 BCCRT 149 https://decisions.civilresolutionbc.ca/crt/crtd/en/item/525448/index.do
- NIST, AI Risk Management Framework https://www.nist.gov/itl/ai-risk-management-framework
- NAIC, Model Bulletin (adopted) https://content.naic.org/sites/default/files/inline-files/2023-12-4%20Model%20Bulletin_Adopted_0.pdf
- EUR-Lex, Regulation (EU) 2022/2554 https://eur-lex.europa.eu/eli/reg/2022/2554/oj; (text read from http://publications.europa.eu/resource/celex/32022R2554;) ; EIOPA https://www.eiopa.europa.eu/digital-operational-resilience-act-dora_en
- EUR-Lex, Regulation (EU) 2024/1689 https://eur-lex.europa.eu/eli/reg/2024/1689/oj; (text read from http://publications.europa.eu/resource/celex/32024R1689)
- Andres Freund, oss-security, backdoor in upstream xz/liblzma leading to ssh server compromise https://www.openwall.com/lists/oss-security/2024/03/29/4
- NIST NVD, CVE-2024-3094 https://nvd.nist.gov/vuln/detail/CVE-2024-3094
- CrowdStrike, External Technical Root Cause Analysis: Channel File 291 https://www.crowdstrike.com/wp-content/uploads/2024/08/Channel-File-291-Incident-Root-Cause-Analysis-08.06.2024.pdf
- The series thread for the invented insurer (illustrative)
- EUR-Lex, Regulation (EU) 2026/1744 https://eur-lex.europa.eu/eli/reg/2026/1744/oj; (text read from the Publications Office copy https://publications.europa.eu/resource/cellar/b459c07f-86fb-11f1-bf5e-01aa75ed71a1.0006.03/DOC_1)
- NIST AI 100-1 https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.100-1.pdf
- NIST, Adversarial Machine Learning: A Taxonomy and Terminology of Attacks and Mitigations https://csrc.nist.gov/pubs/ai/100/2/e2025/final
- METR, Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity https://metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/
- Prime Video Tech (archived copy), as above
- NAIC, Model Bulletin (adopted), as above
- arXiv:2302.06590 https://arxiv.org/abs/2302.06590
- Federal Register, Vol. 86, No. 93, 17 May 2021 https://www.govinfo.gov/content/pkg/FR-2021-05-17/html/2021-10460.htm
- METR https://metr.org/blog/2026-02-24-uplift-update/
- J. Cardena, From Silicon to Screen https://blog.jcardena.com/en/ebooks/from-silicon-to-screen/

Chapter 11. What Returned, and What Did Not Carry Over

- Ariane 501 Inquiry Board report, Paris, 19 July 1996, sections 1.1 and 2.1 https://esamultimedia.esa.int/docs/esa-x-1819eng.pdf
- GAO product page https://www.gao.gov/products/aimd-10.1.14

- GAO/AIMD-10.1.14 https://www.gao.gov/assets/aimd-10.1.14.pdf
- IBM Enterprise COBOL for z/OS Language Reference, "COPY statement" https://www.ibm.com/docs/en/SS6SG3_6.5/lr/ref/rlcdscop.html
- W3C TAG, Extending and Versioning Languages: Terminology, Draft TAG Finding 13 November 2007 https://www.w3.org/2001/tag/doc/versioning-20071113.html
- W3C, Guide to Versioning XML Languages using new XML Schema 1.1 features, Working Draft 20 July 2007 https://www.w3.org/TR/2007/WD-xmlschema-guide2versioning-20070720/
- NVD, CVE-2021-44228 https://nvd.nist.gov/vuln/detail/CVE-2021-44228
- Apache Logging Services, Security https://logging.apache.org/security.html
- IBM Systems Journal, DOI 10.1147/sj.153.0225 https://doi.org/10.1147/sj.153.0225
- M. M. Lehman, Programs, life cycles, and laws of software evolution, Proceedings of the IEEE 68(9), 1980 https://doi.org/10.1109/PROC.1980.11805
- Sun Microsystems Laboratories technical report http://web.archive.org/web/20090717005030/http://research.sun.com/techrep/1994/smli_tr-94-29.pdf
- IEEE Software, DOI 10.1109/52.469757 https://doi.org/10.1109/52.469757
- https://martinfowler.com/bliki/OriginalStranglerFigApplication.html
- University of Cambridge Computer Laboratory, EDSAC 1 and after https://www.cl.cam.ac.uk/events/EDSAC99/history.html
- Communications of the ACM, DOI 10.1145/361598.361623 https://doi.org/10.1145/361598.361623
- ACM Transactions on Computer Systems, DOI 10.1145/2080.357392 https://doi.org/10.1145/2080.357392
- OMG, History of CORBA https://www.omg.org/corba/history_of_corba.htm
- J. H. Saltzer, tribute to F. J. Corbató, MIT https://web.mit.edu/saltzer/www/publications/CorbatoTribute.pdf
- Jim Gray, Why Do Computers Stop and What Can Be Done About It?, Tandem TR 85.7, June 1985, pp. 11 to 15 https://jimgray.azurewebsites.net/papers/TandemTR85.7_WhyDoComputersStop.pdf
- NVD https://nvd.nist.gov/vuln/detail/CVE-2021-45046; https://nvd.nist.gov/vuln/detail/CVE-2021-45105; https://nvd.nist.gov/vuln/detail/CVE-2021-44832; Apache Logging Services, Security https://logging.apache.org/security.html
- NASA, Mars Climate Orbiter Mishap Investigation Board Phase I Report, 10 November 1999, p. 16 https://llis.nasa.gov/llis_lib/pdf/1009464main1_0641-mr.pdf
- Illustrative thread; no external source.
- LinkedIn Engineering (archived copy) https://web.archive.org/web/2014/https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying
- From Tally to Token, published ebook https://blog.jcardena.com

Chapter 12. One Claims System Today: Keep, Wrap or Replace

- IBM Heritage, System/360 https://www.ibm.com/history/system-360
- IBM Archives, System/360 Model 30 and Model 40 product profiles http://web.archive.org/web/20191216083407/https://www.ibm.com/ibm/history/exhibits/mainframe/mainframe_PP2030.html; and http://web.archive.org/web/20191216083408/https://www.ibm.com/ibm/history/exhibits/mainframe/mainframe_PP2040.html
- Martin Fowler, Original Strangler Fig Application https://martinfowler.com/bliki/OriginalStranglerFigApplication.html
- Martin Fowler, Strangler Fig https://martinfowler.com/bliki/StranglerFigApplication.html
- OpenAPI Initiative https://www.openapis.org/; AsyncAPI https://www.asyncapi.com/; CloudEvents https://cloudevents.io/; OpenTelemetry https://opentelemetry.io/; Open Container Initiative https://opencontainers.org/
- ACORD, About https://www.acord.org/about
- US GAO, Information Technology: Federal Agencies Need to Address Aging Legacy Systems https://www.gao.gov/products/gao-16-468
- Illustrative thread; no external source.

Epilogue: Designed to Run, and the Next Binding Constraint

- International Energy Agency, Key Questions on Energy and AI (2026), executive summary https://www.iea.org/reports/key-questions-on-energy-and-ai/executive-summary
- EUR-Lex, Directive (EU) 2024/2853 http://data.europa.eu/eli/dir/2024/2853/oj
- European Parliament Legislative Observatory, summary of the final act, procedure 2022/0302(COD) https://oeil.europarl.europa.eu/oeil/en/document-summary?id=1795969

- Final adopted text, PE-CONS 7/24, 2022/0302(COD) https://data.consilium.europa.eu/doc/document/PE-7-2024-INIT/en/pdf; EUR-Lex http://data.europa.eu/eli/dir/2024/2853/oj
- Ariane 501 Inquiry Board report, 19 July 1996 https://esamultimedia.esa.int/docs/esa-x-1819eng.pdf; NASA, Mars Climate Orbiter Mishap Investigation Board Phase I Report, 10 November 1999 https://llis.nasa.gov/llis_lib/pdf/1009464main1_0641-mr.pdf

Appendix B. Timeline, 1945 to 2026

- IBM Heritage, IBM 650 https://www.ibm.com/history/650
- InformIT (Pearson) https://www.informit.com/store/design-patterns-elements-of-reusable-object-oriented-9780201633610
- LEO Computers Society https://www.leo-computers.org.uk/in-november-2026-leo-will-have-its-75th-birthday/
- OMG, History of CORBA https://www.omg.org/corba/history_of_corba.htm; OMG, CORBA specification https://www.omg.org/spec/CORBA/
- Martin Fowler https://martinfowler.com/bliki/OriginalStranglerFigApplication.html; https://martinfowler.com/bliki/StranglerFigApplication.html
- W3C https://www.w3.org/TR/xmlschema-guide2versioning/; https://www.w3.org/2001/tag/doc/versioning
- US GAO https://www.gao.gov/products/gao-16-468
- European Parliament Legislative Observatory https://oeil.europarl.europa.eu/oeil/en/document-summary?id=1795969; EUR-Lex http://data.europa.eu/eli/dir/2024/2853/oj
- Official Journal text, Publications Office http://publications.europa.eu/resource/celex/32026R1744; EUR-Lex https://eur-lex.europa.eu/eli/reg/2026/1744/oj
- IBM, Accounting Machine Type 407: Preliminary Manual of Information, form 22-5765-0, 1949, Computer History Museum catalog 102724936 (Chapter 3 sources)
- LEO Computers Society https://www.leo-computers.org.uk/?p=3879 Centre for Computing History, citing Ernest Lenaerts's notebooks (Chapter 3 sources)
- IBM, OS/VS VSAM Planning Guide http://www.bitsavers.org/pdf/ibm/370/OS_VS2/Release_1_1972/
- IBM Redbooks, TIPS0925 https://www.redbooks.ibm.com/abstracts/tips0925.html
- Computer History Museum, catalog 102651720 (Chapter 6 sources)
- IBM Heritage, Information Management System https://www.ibm.com/history/information-management-system; IBM z/OS Basic Skills, Introduction to IMS https://www.ibm.com/docs/en/zos-basic-skills?topic=now-introduction-ims
- IBM Support, CICS Transaction Server Release History https://www.ibm.com/support/pages/node/658565; IBM Redbook SG24-7952 https://www.redbooks.ibm.com/abstracts/sg247952.html
- IBM Archives, product profiles http://web.archive.org/web/20191216083407/https://www.ibm.com/ibm/history/exhibits/mainframe/mainframe_PP2030.html; http://web.archive.org/web/20191216083408/https://www.ibm.com/ibm/history/exhibits/mainframe/mainframe_PP2040.html

Appendix C. Glossary

- Stevens, Myers and Constantine, Structured Design, IBM Systems Journal 13(2), 1974 https://doi.org/10.1147/sj.132.0115
- James Lewis and Martin Fowler, Microservices, 25 March 2014 https://martinfowler.com/articles/microservices.html
- W3C, Web Services Description Language (WSDL) 1.1, W3C Note 15 March 2001 https://www.w3.org/TR/2001/NOTE-wsdl-20010315
- Martin Fowler, Original Strangler Fig Application https://martinfowler.com/bliki/OriginalStranglerFigApplication.html
- Saltzer, Reed and Clark, End-to-End Arguments in System Design, ACM TOCS 2(4), November 1984 https://doi.org/10.1145/357401.357402
- Jim Gray, Why Do Computers Stop and What Can Be Done About It?, Tandem TR 85.7, June 1985 https://jimgray.azurewebsites.net/papers/TandemTR85.7_WhyDoComputersStop.pdf
- L. Peter Deutsch, Software Engineering Radio episode 470, July 2021 https://se-radio.net/2021/07/episode-470-l-peter-deutsch-on-the-fallacies-of-distributed-computing/; Sun page archived 1999 https://web.archive.org/web/19991008110904/http://java.sun.com/people/jag/Fallacies.html
- Saltzer and Schroeder, The Protection of Information in Computer Systems, Proceedings of the IEEE 63(9), September 1975 https://doi.org/10.1109/PROC.1975.9939

Appendix D. Platform Mapping, as of November 2026

- Microsoft Learn, "Service Bus queues, topics, and subscriptions" https://learn.microsoft.com/azure/service-bus-messaging/service-bus-queues-topics-subscriptions
- AWS, "What is Amazon Simple Queue Service?" https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/welcome.html
- Google Cloud documentation https://docs.cloud.google.com/tasks/docs/dual-overview; https://docs.cloud.google.com/pubsub/docs/overview
- NATS docs https://docs.nats.io/nats-concepts/jetstream; CNCF https://www.cncf.io/projects/nats/
- Microsoft Learn, "What is Azure Event Hubs?" https://learn.microsoft.com/azure/event-hubs/event-hubs-about
- AWS documentation https://docs.aws.amazon.com/streams/latest/dev/introduction.html; https://docs.aws.amazon.com/msk/latest/developerguide/what-is-msk.html
- Google Cloud documentation https://docs.cloud.google.com/managed-service-for-apache-kafka/docs/overview
- CNCF https://www.cncf.io/projects/strimzi/; Strimzi overview https://strimzi.io/docs/operators/latest/overview
- Debezium https://debezium.io/
- Microsoft Learn https://learn.microsoft.com/azure/event-grid/overview
- AWS https://docs.aws.amazon.com/eventbridge/latest/userguide/eb-what-is.html
- Google Cloud documentation https://docs.cloud.google.com/eventarc/docs
- Knative docs https://knative.dev/docs/eventing/; https://knative.dev/docs/serving/; CNCF https://www.cncf.io/projects/knative/
- Microsoft Learn https://learn.microsoft.com/azure/azure-functions/durable/durable-functions-overview; https://learn.microsoft.com/azure/logic-apps/logic-apps-overview
- AWS https://docs.aws.amazon.com/step-functions/latest/dg/welcome.html
- Google Cloud documentation https://docs.cloud.google.com/workflows/docs/overview
- CNCF https://www.cncf.io/projects/argo/
- Microsoft Learn https://learn.microsoft.com/azure/azure-functions/functions-overview
- AWS https://docs.aws.amazon.com/lambda/latest/dg/welcome.html
- Google Cloud documentation https://docs.cloud.google.com/run/docs/functions/comparison
- Microsoft Learn https://learn.microsoft.com/azure/aks/what-is-aks
- AWS https://docs.aws.amazon.com/eks/latest/userguide/what-is-eks.html
- Google Cloud documentation https://docs.cloud.google.com/kubernetes-engine/docs/concepts/kubernetes-engine-overview
- Kubernetes documentation https://kubernetes.io/docs/concepts/workloads/controllers/deployment/
- Microsoft Learn https://learn.microsoft.com/azure/api-management/api-management-key-concepts
- AWS https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html
- Google Cloud documentation https://docs.cloud.google.com/apigee/docs/api-platform/get-started/what-apigee; https://docs.cloud.google.com/api-gateway/docs/about-api-gateway
- Gateway API releases https://github.com/kubernetes-sigs/gateway-api/releases; docs https://gateway-api.sigs.k8s.io/docs/
- Microsoft Learn https://learn.microsoft.com/azure/aks/istio-about
- AWS, What is AWS App Mesh? https://docs.aws.amazon.com/app-mesh/latest/userguide/what-is-app-mesh.html; https://docs.aws.amazon.com/vpc-lattice/latest/ug/what-is-vpc-lattice.html; https://docs.aws.amazon.com/AmazonECS/latest/developerguide/service-connect.html
- Google Cloud documentation https://docs.cloud.google.com/service-mesh/docs/overview
- CNCF https://www.cncf.io/projects/istio/; https://www.cncf.io/projects/linkerd/
- Microsoft Learn https://learn.microsoft.com/azure/data-factory/concepts-change-data-capture-resource
- AWS https://docs.aws.amazon.com/dms/latest/userguide/CHAP_Task.CDC.html
- Google Cloud documentation https://docs.cloud.google.com/datastream/docs/overview
- Microsoft Learn https://learn.microsoft.com/azure/event-hubs/schema-registry-concepts
- AWS https://docs.aws.amazon.com/glue/latest/dg/schema-registry.html
- Google Cloud documentation https://docs.cloud.google.com/pubsub/docs/schemas
- CNCF https://www.cncf.io/projects/apicurio-registry/
- Microsoft Learn https://learn.microsoft.com/azure/azure-monitor/app/app-insights-overview
- AWS https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/CloudWatch-OpenTelemetry-Sections.html; https://docs.aws.amazon.com/xray/latest/devguide/xray-sdk-migration.html

- Google Cloud documentation https://docs.cloud.google.com/trace/docs/overview; https://docs.cloud.google.com/stackdriver/docs
- CNCF https://www.cncf.io/projects/opentelemetry/; https://www.cncf.io/projects/jaeger/; https://www.cncf.io/projects/prometheus/
- Microsoft Learn https://learn.microsoft.com/azure/key-vault/general/overview
- AWS https://docs.aws.amazon.com/secretsmanager/latest/userguide/intro.html
- Google Cloud documentation https://docs.cloud.google.com/secret-manager/docs/overview
- https://kubernetes.io/docs/concepts/configuration/secret/; CNCF https://www.cncf.io/projects/external-secrets/
- Microsoft Learn https://learn.microsoft.com/azure/governance/policy/overview
- AWS https://docs.aws.amazon.com/verifiedpermissions/latest/userguide/what-is-avp.html
- Google Cloud documentation https://docs.cloud.google.com/load-balancing/docs/auth-policy/auth-policy-overview
- CNCF https://www.cncf.io/projects/open-policy-agent-opa/
- Microsoft Learn, "What is Microsoft Foundry Agent Service?" https://learn.microsoft.com/azure/foundry/agents/overview; Microsoft Azure Blog, Azure at Microsoft Ignite 2025 https://azure.microsoft.com/blog/azure-at-microsoft-ignite-2025-all-the-intelligent-cloud-news-explained/
- AWS https://docs.aws.amazon.com/bedrock-agentcore/latest/devguide/what-is-bedrock-agentcore.html
- Google Cloud release notes https://docs.cloud.google.com/gemini-enterprise-agent-platform/release-notes
- CNCF https://www.cncf.io/projects/kagent/
- IBM Documentation, Mapping the JMS delivery option and message reliability to and from the WebSphere MQ persistence value https://www.ibm.com/support/knowledgecenter/en/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/rjc0014_.html
- AWS, AWS Mainframe Modernization availability change https://docs.aws.amazon.com/m2/latest/userguide/mainframe-modernization-availability-change.html
- Vendor and project pages listed above

THE SERIES

Every Boundary Holds

Every Boundary Holds is a series about software architecture: where each era drew its lines, the contract on every boundary, and how a design runs and proves itself when agents build. This is its first book. Each book stands on its own.

Also by the author:

OTHER SERIES

From Bit to Pixel
Every Pixel Matters

From Silicon to Screen
Every Pixel Matters

The Model as Reader
Every Record Counts

What a Value Means
Every Record Counts

The Contract Decides
GDCD Field Guides

Governing Azure by Contract
GDCD on Azure

From Bronze to Gold
The Medallion Series

From Code to Glass
Every Pixel Matters

From Tally to Token
Every Record Counts

From Warehouse to Lakehouse
Every Record Counts

Data on Every Desk
Every Record Counts

Governing Rust by Contract
GDCD in Code

AI Foundry by Contract
GDCD on Azure

FIELD GUIDES

Ten Engineers, the Output of Thirty
Field guide

The Fifth Mind
Field guide

Every Pixel Matters
Field guide

The Forge and The Universe
Field guide

Who Is the Agent?
Field guide

The Agent Loop
Field guide

The Mesh
Field guide

The Crystal and The Ocean
Field guide

The Story of Azure
Field guide

The Security You Can Watch
Field guide

Juan Cardena

Enterprise AI & Agentic Systems Architect · blog.jcardena.com

Download the latest edition and the blank Decision Card and the platform mapping at blog.jcardena.com/en/ebooks/from-terminal-to-agent.

Every Boundary Holds · Book 1 · From Terminal to Agent.