Architecture decisions are the ones that are expensive to undo

21 October 2026 · 6 min read · blogs.iictp.org/architecture-decisions-expensive-to-undo

Most engineering choices can be reversed in an afternoon. A few cannot: how a system is divided, where its data lives, and which parts must agree with each other in real time. This is the first of five TPBOK domains, Architecture & Systems Design, and it is about making those choices well.

TL;DR The short version

- A few architecture choices are expensive to undo: how a system is divided, where its data lives, and which parts must agree in real time.
- Draw module boundaries around the design decisions most likely to change, and expect your system to mirror how your organisation communicates.
- When a network can fail you must trade consistency against availability, so decide per feature which one you will give up and write it down.

Read online, listen to it, or change the text size: https://blogs.iictp.org/architecture-decisions-expensive-to-undo

Hide the decisions most likely to change

In 1972 David Parnas argued that the way to divide a system into modules is not by the steps of a process but by the design decisions each module hides. If a decision is likely to change, such as a data format or a vendor, only one module should know about it. When that decision changes, one module changes, not forty.

This is still the best test of a boundary. Ask of any proposed split: what is hidden behind it, and how likely is that thing to change?

Separate essential complexity from accidental complexity

Fred Brooks, in “No Silver Bullet”, separated the difficulty that comes from the problem itself (essential) from the difficulty we add through our tools and choices (accidental). Better tools can remove accidental complexity. Nothing removes the essential kind, so a practitioner’s job is to avoid adding to it. A design that needs three services, two queues and a cache to do what one function could do has added accidental complexity.

Your system will look like your organisation

Melvin Conway observed in 1968 that organisations design systems that mirror their own communication structure. If three teams must agree on every change, you will end up with three tightly bound components. Practitioners use this deliberately: change the team boundaries first if you want different system boundaries.

Distributed systems force trade-offs you cannot avoid

When a network can fail, a system cannot be both fully consistent and fully available. Eric Brewer put this forward as a conjecture, and Seth Gilbert and Nancy Lynch proved a formal version in 2002. The practical lesson is not to memorise the theorem. It is to decide, per feature, which you would rather give up under failure, and to say so in writing.

A short checklist before you split a system

1. What decision does this boundary hide, and how likely is it to change?
2. Which team owns each side, and can they release independently?
3. What happens to each side when the other is down?
4. What will it cost to run, monitor and debug as two things instead of one?
5. How would we undo this in a year if we were wrong?

> Anti-pattern to watch for: the distributed monolith. The system is split into services, but they must all be deployed together and call each other in long synchronous chains. You pay the cost of distribution and keep the coupling.

How this appears in IICTP exams

Architecture & Systems Design carries 30% of the CTP exam and 24% of the ATP exam. The ATP asks whether you can read a baseline architecture and see how its parts depend on each other. The CTP gives scenarios, such as whether and how to split a system under load, and asks for the best decision, not a perfect one. See the exam facts for every credential's share.

Record your decisions as you make them. A short written note, with the options you rejected, is worth more than a diagram nobody updates.

Sources and further reading

1. Brooks, F. P. (1987). "No Silver Bullet: Essence and Accidents of Software Engineering." IEEE Computer 20(4).
2. Conway, M. E. (1968). "How Do Committees Invent?" Datamation 14(4).
3. Evans, E. (2003). Domain-Driven Design. Addison-Wesley.
4. Gilbert, S. and Lynch, N. (2002). "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services." ACM SIGACT News 33(2).
5. Nygard, M. (2011). "Documenting Architecture Decisions." Cognitect blog.
6. Parnas, D. L. (1972). "On the Criteria To Be Used in Decomposing Systems into Modules." Communications of the ACM 15(12).

Published by IICTP, an independent technical credential body, on 21 October 2026. The current version of this post is at https://blogs.iictp.org/architecture-decisions-expensive-to-undo. More guides: blogs.iictp.org.