



STRATEGIC ESSAY

Editorial date: 18 October 2026

The Engineering That Outlives Its Makers

Why could we repair a computer 24 billion kilometres away, while owners of modern cars had to form an organisation to fix what they had bought?

The Reproduction Boundary: Engineering Saudi Technology to Outlive Its Original Maker

Opening: Technological sovereignty is not simply a matter of where a server resides or who holds the invoice for the hardware. This essay offers an operational definition that can be tested: how far can an organisation reproduce, run, repair and verify its software if the company that made it disappears? The inquiry brings together the repair of Voyager 1, communities of Fisker Ocean car owners and VanMoof bicycle owners, and the science of Reproducible Builds. It proposes a future Saudi industry that sells more than technology: it sells the ability to pass that technology on to whoever will need to repair it years later.

Evidence boundary: The events involving the spacecraft and the vehicles, and the technical decisions discussed below, are dated and supported by primary statements, with explicit limits on what they establish. The “Reproduction Boundary” and the “Technical Succession Trial” are analytical proposals developed in this essay, not adopted Saudi standards. We ran a real experiment that built a small package twice and compared the resulting hashes, along with a deterministic examination of a synthetic dependency graph. The results contain no data from Saudi systems and no probability estimates for conflict.

1. It was within reach of repair, yet no one could reach it

On 20 April 2024, engineers at the Jet Propulsion Laboratory (JPL) confirmed that Voyager 1's computer had resumed sending readable engineering data after months of returning signals whose contents they could not interpret. The spacecraft was then more than **24 billion kilometres** from Earth. A signal took about **22 and a half hours** to reach it, and the same time to return. [S01](#)

No one could dispatch a technician, replace a failed chip or buy a three-year extended-service warranty. Diagnosis showed that a chip in the Flight Data Subsystem (FDS) memory was no longer working properly. The team chose to relocate parts of the software code elsewhere in memory. No single location had enough space for the whole damaged section, so its parts had to be distributed and the references between them adjusted. The engineers first tested the section that returned engineering data and announced its success before restoring the remaining science data in later stages. [S01S02](#)

This is not a story about an immortal machine. The spacecraft is old, its capabilities are diminishing, and repairing it demanded rare expertise and exceptional communications infrastructure. Nor is it an experiment proving that a space project can be independent of its suppliers. But it reveals an engineering truth worth inscribing above the doors of technology laboratories: **our ability to repair something is not directly proportional to how close it is to us.**

That same year, some Fisker Ocean electric-car owners faced a different kind of distance. The car was parked outside the house; its physical components were within reach. But its manufacturer entered bankruptcy proceedings in 2024, exposing difficult questions about the digital services and technical tools on which parts of vehicle maintenance and feature operation depended. On **26 July 2024**, the Fisker Owners Association (FOA) filed a statement in the US bankruptcy case explaining its need for essential cloud services to continue and for access to the vehicle software architecture, technician tools and spare parts. This was a statement by an interested party, not a court finding that all those services had already stopped or that the cars had become undriveable. [S08](#)

The owners did something that should command the attention of anyone buying thousands of processors or signing a contract for a national system. They did not merely create a complaints forum. They organised an association and engaged engineers. Within that owners' ecosystem, OceanLink Pro appeared as an alternative application enabling some diagnostics, control and connectivity within the limits of the available hardware, software and rights. The developer and provider listed by the App Store is Undercurrent Inc.; the product description identifies it as the official app for FOA members. The listing describes claimed functions, but it should not be turned into independent proof that every car recovered its full functionality or that every update was safe. [S09](#)

We are left with a paradox that conventional technology accounting struggles to classify: **a spacecraft beyond physical reach remained repairable; a car within physical reach required its owners to become something resembling the company that made it.**

The distance here is not measured in kilometres. It is the distance between owning a product and owning the knowledge and tools needed to recreate a function it has lost.

2. The lifespan of the maker, the thing and the language between them

Consider a third story, not because it mirrors the car's, but because it exposes a different timescale in technology.

On 17 July 2023, the District Court of Amsterdam declared Dutch entities belonging to electric-bicycle maker VanMoof bankrupt. A published report from the bankruptcy trustees records the court's decision, so there is no need to rely on the rumours that preceded the announcement. [S03](#)

During those anxious days, a competitor, Cowboy, released an application called Bikey. Its launch description foregrounded local-key support for VanMoof S3/X3 bicycles, while the version history later recorded support for A5/S5 models. Users could save a digital key on their phone and use it if the original company's service stopped. The listing does not establish complete long-term support for every model, prove that every bicycle would have stopped working if the servers failed, or show that the application solved every maintenance need. [S04](#)

Yet the company did not vanish from the earth. A new owner acquired the brand's assets in 2023, and in February 2024 VanMoof published an archive of repair manuals and technical drawings for some older models. Its announcement explained that material previously limited to certain workshops had become available and that some files had needed translation from Chinese. This was a step in moving knowledge beyond the factory walls to whoever would repair the bicycle later. It was not evidence that every software code, authentication key or spare part had become public. [S05S06](#)

Then, in 2026, another kind of loss appeared, one that requires no corporate bankruptcy: **protocol ageing.**

In its guidance on the 2G Network Phase-out, VanMoof explained that theft tracking and remote connectivity in older models would be affected by the decline of the networks on which they relied. It set a deadline before **1 April 2026** for customers who wanted to activate a Ride Pro subscription and retain a particular tracking service through an arrangement

negotiated with network providers. Crucially, it stated explicitly that **riding, locking and other independent functions would continue to work**. The event should not be reported as “the bicycles stopped working”. [S07](#)

There are therefore three lifespans, and they need not coincide: the life of the company selling the product; the life of the material object a person sits on or drives; and the life of the technical language through which the product communicates with the world, from a radio frequency to an encryption suite or an Application Programming Interface (API).

The bicycle may outlive the bankruptcy of its maker. Its company may return to the market while a communications network embedded in an older model’s design disappears. A buyer may own every screw in a device yet remain unable to reproduce a single software component designed when its compiler was available and its licence still valid.

The subject is therefore deeper than “supplier dependence”. We are asking about **the heritability of technology**: can a function pass to another generation of engineers, hardware and software as a musical instrument passes to a player who knows how to restring it?

3. The essay’s concept: the Reproduction Boundary

I propose the term **Reproduction Boundary** for the last step that the owner of a digital service can complete independently, lawfully and verifiably before it requires an asset, decision or body of knowledge held exclusively by another provider.

Consider the path software follows before it reaches a user:

Source Code ← Toolchain ← Dependencies ← Binary Artifact ← Release Signing ← Runtime and Hardware ← Data and Configuration ← Acceptance Tests.

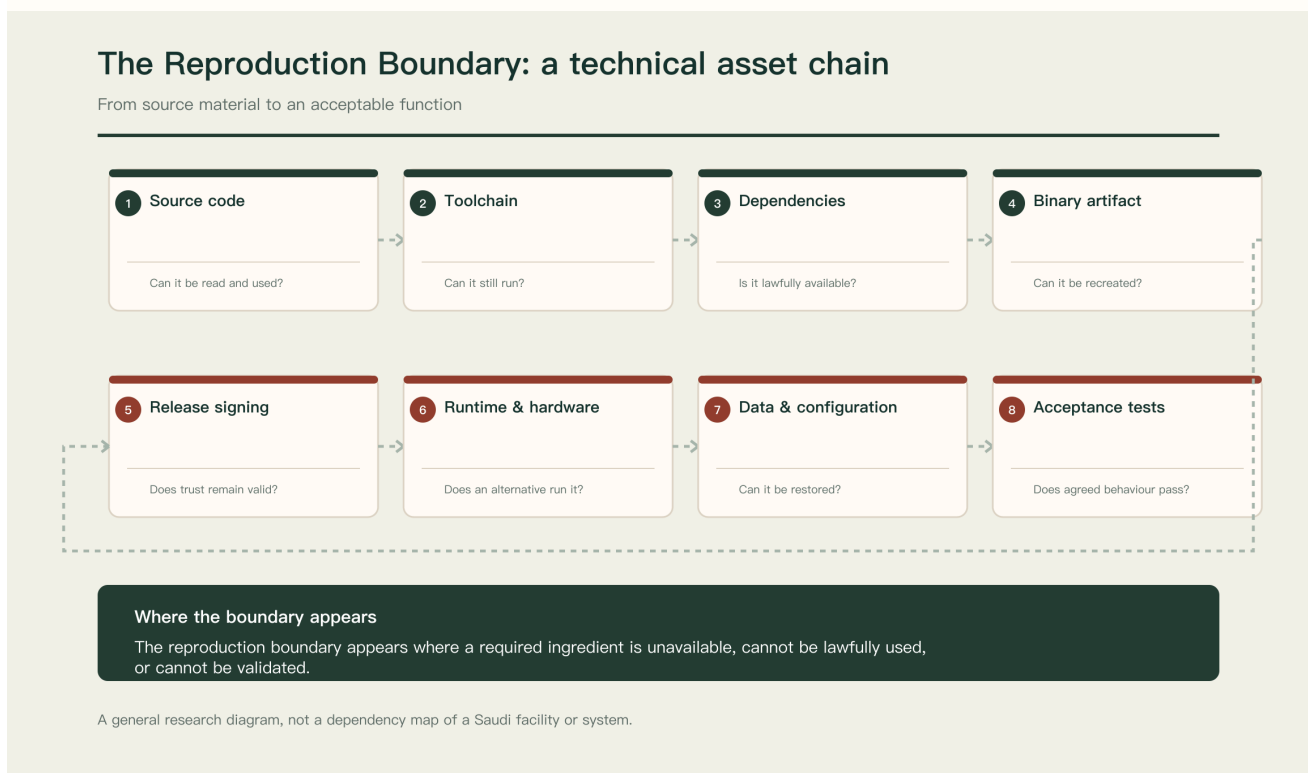
This is not a one-way chain in every product. Dependencies branch, some systems use pre-built images, and some stages run concurrently. But it is enough to expose a common illusion: an organisation may own the source code, retain server images and possess the data, yet still be unable to update the software because release signing remains with the manufacturer, because its Software Development Kit (SDK) is no longer available, or because a hardware interface accepts only Signed Firmware supplied by another party.

The opposite can also happen: a binary may continue to run for years while no one can safely modify and rebuild it. “It works now” and “I can produce a new instance of it” are different claims.

Four actions must be distinguished rather than counted indiscriminately as ownership:

Action	The real test	The illusion it may be confused with
Preserve	Have the required assets remained documented, readable and lawful to retain?	Possessing an archive of unknown contents
Rebuild	Can we turn the source into an independent binary that matches the target, or document its differences?	Owning the source alone
Repair	Can we change faulty behaviour and pass known tests?	Restarting the old version
Port/Replace	Can we preserve an essential function on a different platform?	Moving data without restoring the function

An organisation may pass the first two and fail the third. It may also run a replacement with fewer features, one that does not match the old product but meets agreed core needs. That cannot be counted as complete technical equivalence. It is a **deliberate reconstruction of function**.



Conceptual diagram of the Reproduction Boundary and the stages used to test technology

Figure 1 — The proposed Reproduction Boundary: which build assets do we control, and which assets remain subject to another party’s availability, lawful-use terms or signing integrity? This is a general research design, not a dependency map of a Saudi facility.

4. You may hold the code and still lack the key that turns it into a program

More than forty years ago, Ken Thompson's celebrated lecture "Reflections on Trusting Trust" posed a problem that unsettles easy confidence in reading source code: the Compiler itself can carry behaviour that never appears in the source it is meant to translate. [S16](#)

We need not retell the attack scenario here. Its intellectual consequence is enough for our purpose: **the source alone does not explain how the version that actually runs came into being.** Behind it lies a history of build tools, library versions, environment variables, assembly rules and, at times, defects retained so long that they become part of expected behaviour.

"We kept the code" therefore differs from "we rebuilt the application independently and proved that the output matches the intended version". The Reproducible Builds project defines reproducibility through the ability of parties using the same inputs, environment and instructions to obtain Bit-for-Bit identical outputs. A Cryptographic Hash can then compare the outputs, but it does not by itself prove that the source is correct, the program is safe or the right to run it is available. [S11](#)

Three distinct technical concepts are useful here. They are not three different labels for printing the same certificate of readiness:

1. A **Software Bill of Materials (SBOM)**, including the SPDX format, can tell us a great deal about what goes into a program, its relationships and its licences. On its own, it cannot guarantee that every build tool, deployment secret or hardware component is available. [S13](#)
2. **Build Provenance** under the Supply-chain Levels for Software Artifacts (SLSA) specification helps document who built the product, how it was built and under what assurances. Documentation does not automatically confer an independent right to use every component or the ability to alter every device. The current version reviewed here is SLSA 1.2, not an older release applied by default. [S12](#)
3. **The Update Framework (TUF)** provides mechanisms for verifying the authenticity of updates and protecting delivery paths against some key-related and rollback risks. It is not permission to bypass protections imposed by a manufacturer. It is a model showing that safe update delivery depends on a trust architecture whose own continuity must be designed. [S14](#)

Organisations mature when they keep these technologies distinct. There is something absurd about a vault containing a thousand seals of integrity while the workshop cannot make the object those seals certify. The joke ends when that object is a service on which a person depends or a productive asset on which an economy runs.

5. Software memory: where does knowledge hide when the disk does not preserve it?

If the computer knows everything, why did repairing Voyager 1 require months of meticulous human work? Because code, memory and reports do not remove the need for **Tacit**

Operational Knowledge: an understanding of what can be moved, what must not be broken and what each change means in a system whose parts have evolved across decades.

Some knowledge lives in files, some in minds, and some in the behaviour of devices that do not act quite as their designers expected. Copying a Git Repository from one server to another is therefore insufficient. An organisation should preserve a description of the build environment; a copy of licensed dependencies; baseline behavioural comparisons; training material that acknowledges what is no longer understood; and the ability to trial a repair without placing a live service at risk.

One global project illuminates part of the answer: Software Heritage, which seeks to preserve public source code, its development history and the ability to retrieve it. It is an important example of preserving source across time, not a promise that every archived program can run on any future device or a licence to redistribute closed material. [S15](#)

For an old information system, the rarest asset may be neither processor nor disk, but a simple test file showing how the system calculated a sensitive number twenty years ago. That file turns a disagreement between two teams from competing opinions into a result that can be examined. When no one remains who knows what the number meant, changing the software becomes a linguistic gamble: we know how the word was written, but not what it meant in its contract with reality.

6. A practical test: reproducing a package rather than remembering its name

To keep the thesis from becoming an essay about the beauty of independence, we carried out two bounded experiments, with their assumptions stated, in the scientific companion.

In the first, we built a small demonstration service in Python, containing a synthetic readout function and two output tests. We then created two separate packages in different paths on disk without an external connection. We fixed the file order, timestamp and package metadata. **The bytes and the two SHA-256 hashes matched exactly** across the two builds, and both simple tests passed. This is a narrow laboratory success. It does not mean that the compiler is independent of its maker, that a third party has verified the results or that we rebuilt a Saudi program. [MODEL]

The second experiment uses an abstract graph of **26 nodes**, including source files, data, build tools, activation rights, signing, hardware and runnable alternatives. It then examines four paths for restoring a synthetic function under the assumption that the supplier disappears. The number of nodes says nothing about the depth of a real system. [MODEL]

Hypothetical restoration route	Primitive inputs unavailable after the supplier disappears	Can the route be completed under the model?
Preserved Golden Image	3	Impossible without authorised alternatives
Source held by an independent party (Source Escrow)	5	Impossible while original tools and runtime factors remain unavailable
Source and local build tools	3	Impossible while activation, signing and hardware requirements remain with the supplier
Portable functional core (Portable Minimum)	0	Possible assuming the tests, rights and alternatives are sound

The final route did not retain every feature in the hypothetical system; we assumed that six functions out of eight were preserved. These figures do not describe Saudi performance, time or probability. They are a **logical test of the conditions under which reproduction is possible**, not an attack simulation or a report about a company. The research companion describes the dependency model's architecture, and the evidence package presents its eight-case results table. [MODEL]

Four recovery recipes when the supplier is absent

Synthetic-model results reported in the supplied verification table

Recovery route	Unavailable primitive roots	Route completion	Assumed functions of 8
Stored executable image	3	Blocked	8/8 Conditional
Source escrow	5	Blocked	8/8 Conditional
Source & local tools	3	Blocked	8/8 Conditional
Portable functional core	0	Feasible in real	6/8

Results from a 26-node synthetic model; 0 means no requirement remains outside the stated recipe in that model.

Comparison of missing links across four hypothetical build recipes

Figure 2 — Results computed from a hypothetical graph: the number of unavailable primitive inputs for each route, comparing complete functions conditional on an external supplier with a narrower alternative core. Zero does not mean the absence of real-world risk; it means only that the synthetic graph leaves no requirement outside the stated recipe.

Preserving source may appear weaker than preserving an image: five elements are missing instead of three. That is true **only under the experiment's two recipes**, not as a general rule. Source creates broader possibilities for future repair, but reaching them requires tools and build packages that a frozen version may not require merely to run. Anyone who fails to distinguish the horizon of immediate operation from the horizon of modification will mistake these numbers for a final judgement on the value of source.

More importantly, a system may possess **99 files out of 100** while the sole missing component prevents a new release. The calculation is not the proportion we have. It is whether there is a complete, valid path from materials to the required function.

7. In the age of AI, inheritance requires more than a code file

Move from a bicycle needing one part to an Artificial Intelligence (AI) system that an organisation can run today, only for one of its processor or software-tool suppliers to disappear years later.

The organisation might say: we kept the Model Weights. Good. What about the Tokenizer, architecture definition, numerical-precision settings, Custom Operators, Acceleration Libraries, Inference Scheduler, safety criteria and evaluation data that reveal regression after a port?

Weights without the correct way to interpret them resemble a musical instrument with every string present, but with no record of their length, tension or the scale the instrument was built to play.

Then comes the economics of migration. Inference results, speed or power consumption may change when workloads move from one acceleration stack to another. It is not enough for the program to open the model file; it must deliver agreed behaviour at acceptable cost and latency. The rights permitting modification and redistribution must be tested, not inferred from the presence of the files.

Here **Portability** becomes a capability that must be paid for. It requires considered abstractions, acceptance tests and sometimes compromises in performance or features. A specialised platform may be more efficient under normal conditions, so independence must not become a synonym for abandoning every advanced global technology. The scientific decision is to compare the **return on specialisation** with the cost of allowing a program to age with its sole provider.

Imagine a Saudi laboratory developing an AI product. Its strongest strategic move may be more than buying additional capacity. It may be acquiring the ability to move a defined model to a second execution path, rerun a core test suite, and explain changes in quality, latency and cost. This does not claim that porting is easy or always possible. It states precisely what should be tested before the first path becomes a fate that cannot be changed.

8. Saudi Arabia: from code repository to reproduction factory

The strategic contribution here must be practical. The Digital Government Authority (DGA) has an announced Government Cloud Software Repository (GPaaS) programme focused on making free and open-source government code available and enabling public bodies to develop and exchange it. Saudi entities including the Real Estate General Authority (REGA) present this direction as a way to support reuse and reduce dependence on some suppliers. This is a published institutional fact. It does not establish that every government service can be rebuilt outside its current provider. [S17S18](#)

The repository is a necessary step along some paths. It is not the end of the test.

If a Saudi technology organisation wanted to turn this direction into a deeper industry, it could begin with what I call a **Technical Succession Trial**, rather than a compliance study with many headings:

Day one: A team that did not build the product opens an asset package isolated from the supplier's servers and establishes what it can obtain in the form of source, dependencies, licence terms, tools and a documented environment.

Day two: It attempts to build a binary; it compares its hash when equivalence is the proper criterion, and records the causes of any difference when the build contains non-deterministic factors.

Day three: It runs a test version using synthetic or properly authorised anonymised data, changes one small function, and reruns the acceptance, security and performance tests.

Day four: It attempts to run a defined core on an alternative that is technically and contractually permitted, without disabling hardware protections or imitating another party's keys.

Day five: It records the functions that remained, what could not be restored, the cost of repair, and who has the right to decide when results differ. These days are an illustrative teaching sequence, not a promise that a complex system can pass the trial in five days.

If the trial fails, the reason must be identified. Is it a licence, a missing tool, a software dependency, a skills gap or a test no one knows how to interpret? Each calls for a different investment. The answer may be a new contract, an authorised Package Mirror, behavioural tests, a Compiler Engineering team or a completely different architecture. The word "localisation" explains none of these steps, even if it can sometimes describe a broader policy or economic goal.

9. Testing the strongest objection: does everything deserve to be born twice?

No. For many products, a specialist supplier may be faster, more efficient and more secure than an attempt to recreate every component internally. Rebuilding a complex platform locally may raise costs, slow security fixes and create a local monopoly supplier more fragile than global competition. In some sectors, contractual rights or safety requirements preclude replacing sensitive software with an uncertified alternative.

Open-source software may also be poorly maintained, and a project may possess every line of source while lacking engineers who understand it. No equation tells us that local is safe, open is independent or distributed is always better.

What, then, is the wise investment? **Selective Reproducibility.**

Functions should be classified by the real return on being able to repair or port them: how difficult the supplier is to replace; how long the service retains its value; the harm caused when support ends; the scarcity of the relevant skill; the number of alternative platforms; the cost of testing; and whether replacing the product would be faster than repairing the old one. In some cases, the rational choice is to buy the product as it is and prepare an exit plan. In others, it should not be bought without early evidence that its functional core can be run independently.

The distinction is not made by sentiment towards a local or foreign supplier. It is made by **technical tests and economic comparisons that change from project to project.**

10. A market we have not yet learned to price

In a world where software outlives some of its companies and communications protocols can age before the hardware reaches the end of its useful life, demand may emerge for an industry closer to **Evolutionary Maintenance** than to the backup business.

Its possible products are not another dashboard glowing green. They are tools that automatically rebuild old releases inside isolated environments; laboratories that revive legacy code; translators for abandoned interfaces; suites that run behavioural tests across different generations of hardware; independent verification services that connect a binary to its source and build environment; and open components that preserve the minimum functionality of a device or service after the supplier stops maintaining it.

As part of its investment in computing, digital services and AI, Saudi Arabia could examine whether this kind of technology might be exported to organisations that do not want their software to become a museum piece when a contract ends. But an industry is not born from the title of a research centre. It needs paid challenges from customers, authorised test data, clear distribution rights, and comparisons between the economics of repairing an original and replacing it.

Foresight to 2035 thus becomes a market question rather than a national slogan: **who will sell the ability to keep other people's products alive after their first generation ends?**

The winner could be a small company that has mastered migration from an old protocol, a team that has built an auditable compiler, or a laboratory that has rehabilitated an AI model on a second platform without destroying its quality. No one can declare them inevitable winners now, but the mechanism of value is clear enough to design a market experiment that measures it.

11. Conclusion: the inheritance that cannot be put in a box

When Voyager 1's engineers repaired old code at a distance that prevents a signal from sustaining an ordinary conversation, they overcame more than geography. They had enough system description, testing infrastructure, expertise and authority to modify and transmit the program to build a path towards a repair that could not be reached physically. S01

When Fisker Ocean owners needed to create an engineering association and maintenance initiatives after the manufacturer faltered, the problem was not an absence of technology in the cars. They were full of it, but part of their ability to evolve remained tied to an economic institution that was no longer what it had been. S08S09

From this invisible distance emerges this essay's thesis: **a technology product is a consumable asset; the ability to reproduce, verify and repair it after its maker has gone is an independent economic capability.**

Building that capability may cost more than some products justify. A test may reveal that the original supplier is better than any alternative. But it is useful to know that before the day the repair becomes necessary.

A machine has two dates: the day it was made, and the first day its owner needs to repair something its maker cannot. Between them, a country can learn a rare technical art: **to buy what serves it today while retaining the path that will let it understand and recreate what it needs tomorrow.**

Inference notes

1. Neither NASA nor Fisker establishes an equivalent weakness in Saudi facilities. The former shows a successful software repair under extreme constraints; the latter records a social and practical effort to transfer part of maintenance to users.
2. The 2023 bankruptcy of VanMoof entities does not mean the brand has now disappeared. New ownership followed, along with a maintenance archive. The 2026 2G phase-out **did not stop the bicycles from being ridden**; its stated effect is limited to theft tracking and remote connectivity for certain models.
3. The repeated-build experiment establishes that a small synthetic package matched in the stated environment; it is not independent certification of a production package. The graph examination supplies no probabilities, recovery hours or national score for technological independence.
4. Independent building or porting does not grant a right to bypass licensing, signing or vehicle-safety restrictions. Any practical experiment on a real system requires proper

authority, testing and technical safeguards.

5. Detailed source roles, event and publication dates, and inferential limits are collected in the source register (`source_register.csv`) and news casebook (`news_casebook.md`) within the evidence package. The proposed “Reproduction Boundary” is the author’s formulation.

Selected sources

- National Aeronautics and Space Administration (NASA), *Voyager 1 Resumes Sending Engineering Updates to Earth*, 22 April 2024. [S01](#)
- District Court of Amsterdam and VanMoof bankruptcy trustees, public bankruptcy report, June 2024, recording the event of 17 July 2023. [S03](#)
- Cowboy, original Bikey app listing, July 2023. [S04](#)
- VanMoof, maintenance-archive announcement of 27 February 2024 and 2G network updates for 2026. [S06S07](#)
- Fisker Owners Association statement filed in the 2024 bankruptcy case, and the OceanLink Pro listing by developer/provider Undercurrent Inc., whose description identifies it as the official app for FOA members. [S08S09](#)
- Reproducible Builds project, technical definition of identical builds. [S11](#)
- SLSA 1.2 specification, approved on 24 November 2025 and current at the evidence cut-off. [S12](#)
- Digital Government Authority, government code-deposit programme. [S17](#)