



RESEARCH COMPANION

Editorial date: 18 October 2026

The Reproduction Boundary: A Testable Model of Supplier- Independent Technical Recoverability

The Reproduction Boundary: A Testable Model of Supplier-Independent Technical Recoverability

Type of work: A Conceptual Research Paper containing two bounded, genuinely executed computational experiments: a Deterministic Artifact Rebuild of a small software package performed twice, and a Supplier-Loss Dependency Closure check over synthetic dependency nodes. It contains no production samples, national estimates or probabilistic forecasts.

Completion status: The defined evidence below has been reviewed; the conceptual model has been designed; the experimental software has been executed and its outputs checked; and the plan for a future field study has been completed. No field comparison, experiment on Saudi services or empirical financial-feasibility study has been conducted.

Evidence cut-off: 18 October 2026. Conditional application horizon: 2026–2035.

Arabic abstract

Problem: Ownership of code, data and physical infrastructure receives considerable attention in the literature on technological independence. Yet these indicators are insufficient to establish whether an organisation can reproduce or repair a functioning service when tools, licences, signatures or knowledge from its original supplier become unavailable. This paper proposes the **Reproduction Boundary**: the final documented transition that a service owner

can complete from source assets to an acceptable, safe function without an indispensable dependency on an inaccessible asset. It is a property of a function, path and context, not an absolute rank assigned to a state.

Evidence: The 2024 repair of Voyager 1 software, maintenance pathways that emerged for Fisker Ocean vehicles after the manufacturer's bankruptcy, and the transfer of knowledge around VanMoof from 2023 to 2026 provide distinct cases for separating technical capability from geography, asset ownership and rights. Reproducible Builds, SLSA 1.2, SPDX and The Update Framework (TUF) offer partial mechanisms for preserving information, establishing provenance or securing releases. None is sufficient on its own to guarantee independent restoration.

Method executed: A small demonstration program in Python and two behavioural tests were created, and two packages were built in separate temporary paths with fixed inputs, file order and timestamps. The two packages matched byte for byte and produced the same SHA-256 digest. A synthetic graph of **26 nodes**, covering local assets, supplier requirements and derivable components, was also examined. **Eight restoration-route cases** were executed: four with the supplier available and four with it absent. Under the latter condition, the archived-image, source-escrow, and source-with-local-build-tools routes lacked **3, 5 and 3** primitive inputs respectively. An assumed portable-core route completed with no missing inputs while retaining **six of eight functions**, without any claim that it fully matched the original.

Conclusion: The examination offers a bounded computational demonstration that preserving many files, or making source available, does not entail the existence of a complete route to restore a defined function when the supplier is absent. It does not establish that the proposed indicators predict Saudi outcomes, nor does it estimate market size or restoration time. We therefore propose an independent, multi-product experiment comparing four tests: preservation of material, rebuilding, behavioural repair and authorised functional porting. A plan for testing and falsification, safety boundaries and implementation costs is included.

Keywords: Technical Independence; Reproduction Boundary; Functional Recoverability; Reproducible Builds; Software Bill of Materials (SBOM); Build Chain Assets; Supplier Dependence; Dependency Closure; Software Inheritability.

English abstract

Question. Does owning a service's data, source code, or compiled image establish the practical ability to rebuild, modify, validate, or replace its essential function when the original supplier is unavailable? We introduce a proposed *Reproduction Boundary*: the last independently

executable and verifiable transition within a service's source-to-operation chain, conditional on specified functionality, lawful access, build tools, cryptographic trust, hardware interfaces, skills, and testing.

Completed work. An original small Python demonstration service was packaged twice at separate temporary locations using identical explicitly fixed archive metadata. Its two SHA-256 digests and raw bytes matched, and two basic functional checks passed. An entirely synthetic 26-node dependency graph and four alternative restoration targets were evaluated both with and without vendor primitives, producing eight deterministic route checks. Under the stipulated supplier-absent condition, the archived-image, source-escrow and local-toolchain routes had respectively 3, 5 and 3 unavailable primitive prerequisites and could not complete. The assumed portable-minimum route had none but retained only six of eight stipulated features. These results are properties of an invented graph and toy package; they are **not supplier-outage probabilities, measured Saudi systems, real security certifications, independent reproductions of production software, or restoration-time estimates.**

Contribution and next study. The framework separates byte-level reproducibility, behavioral acceptability, authorization to deploy, and economic attractiveness. It identifies the data and authorized technical experiments needed to learn whether an independently exercised reconstruction capability predicts shorter or cheaper vendor-exit transitions than inventories, escrow commitments, or system redundancy alone. Proposed falsification and economic decision criteria are explicit; no causal estimate or validated national index is claimed.

Keywords: sovereign recoverability; independent rebuild; software supply chain; hermetic build; functional equivalence; digital maintenance; supplier disappearance; reproducible software; Saudi Arabia.

1. Scientific question and unit of analysis

The central question is: **what conditions enable an organisation to reproduce a defined operational function lawfully, technically and at a demonstrable level of quality without the presence of the original supplier providing some of its tools or operating rights?**

The unit of analysis is not the “state” as a whole, the “data centre”, or an “application” treated as a commercial name. It is a **defined function or bundle of functions** in a particular version of a product, on a known platform, under an authorised supplier-loss scenario, and against a stated acceptance criterion. A simple reporting service may be rebuilt while a component dependent on hardware or a licensed model cannot. A composite measure that erases this distinction cannot settle the matter.

“Digital sovereignty” can have legal and political dimensions that do not all fall within engineering design. This paper therefore separates three questions:

1. **Does the material exist?** Data, source, binaries, documentation, operational images, packages and definitions.
2. **Is the path available and valid?** Tools, usage rights, lawfully controlled signing keys, replaceable hardware, behavioural tests and human expertise.
3. **Is investment in that path worthwhile?** The cost of retaining and renewing the capability, the cost of redevelopment or buying a replacement, the expected harm from supplier absence, and the product's useful life.

These are not maturity levels awarded by intuition. They are different tests with different denominators.

1.1 What could change the answer?

If an authorised study finds that keeping a copy of source does not materially improve an independent team's ability to build a valid release when the supplier is absent, development contracts should change. Attention should move from purchasing source alone towards verifiable retention of the build environment and tests, and perhaps towards developing a portable core. Conversely, if comparative studies show that independent repositories increase failure and cost without improving restoration speed for a class of services, there is no basis for applying them indiscriminately.

1.2 Proposed hypotheses

- **H1 (path completeness):** In products with licensed or hard-to-replace dependencies, passing an actual rebuild test predicts porting and repair capability more clearly than merely possessing a copy of source, after controlling for product complexity, size and team.
- **H2 (functional boundary):** The likelihood of obtaining an acceptable alternative rises when core functions are distinguished from secondary features before the supplier disappears, though this may reduce the number of available features.
- **H3 (asset validity over time):** Preserving source becomes less effective unless mechanisms also keep packages, compiler infrastructure and hardware interfaces current as they age.
- **H4 (conditional return):** Not every service justifies the cost of independent reproduction; value interacts with the service's expected life, degree of customisation, availability of market alternatives and cost of error.

Strong counter-hypothesis: Buying a modern replacement and redesigning processes may be more effective than rebuilding a legacy system. Managed commercial dependencies from a provider with a large customer base may be less risky than a poorly maintained local copy. The experiment should admit both possibilities rather than exclude them as political objections.

2. Review of real-world evidence, separated from inference

2.1 Voyager 1: extreme physical distance, accessible epistemic paths to repair

On 22 April 2024, the US National Aeronautics and Space Administration (NASA) announced that intelligible engineering data had resumed after engineers diagnosed a fault in part of the Flight Data Subsystem (FDS) memory and redistributed a section of code and its instructions elsewhere. Voyager 1 was more than 24 billion kilometres away, with one-way signal travel time of about 22.5 hours. The first observable operational success occurred on 20 April. NASA later announced the return of science data from two instruments in May; the two stages should not be collapsed into a claim of “full restoration” in April. [S01S02](#)

Supported observation: A partial remote software repair succeeded even though the chip could not be repaired physically. **Plausible mechanism:** knowledge of how to interpret the memory, code and internal references, together with authorised means to transmit instructions and receive evidence of validation. **What the event does not establish:** a successful commercial exit agreement, build independence of another spacecraft from its suppliers, or an ordinary technical cost transferable to government services.

2.2 Fisker Ocean: the owner becomes part of the maintenance industry

Car manufacturer Fisker entered bankruptcy proceedings in 2024. In a statement filed in the bankruptcy case on 26 July 2024, the Fisker Owners Association (FOA) explained that continuity required access to code, maintenance tools, cloud services, parts and vehicle communications. It did so as a party seeking an agreement, not a neutral evaluator of outcomes. The App Store listing establishes the existence of OceanLink Pro and names Undercurrent Inc. as its developer and provider; the product description identifies it as the official app for FOA members. Its claimed functions vary according to vehicle hardware and software version. Neither the filing nor the product page establishes that every car stopped working or that all its functions became independent. [S08S09S10](#)

Supported observation: A maintenance and software initiative emerged outside the original manufacturer, and a tool became available from a developer claiming to represent owners. **Plausible interpretation:** limited knowledge distributed among users and engineers was converted into an organised repair capability. **Limit on transfer:** association documents come from an interested party; app claims are not independent tests of every vehicle; some features require specialised equipment; and restoration does not mean that every fault or update right has been resolved. The safety of alternatives should be tested empirically by licensed specialists. We provide no means of accessing vehicle systems or bypassing their safety controls.

2.3 VanMoof: three different shocks and a composite technical lifespan

An original report published by VanMoof's bankruptcy trustees establishes that the District Court of Amsterdam declared its Dutch entities bankrupt on 17 July 2023. During those days, competitor Cowboy published the Bikey application. Its launch description foregrounded local-key support for S3/X3 bicycles, while the version history later recorded A5/S5 support. The listing does not establish complete long-term support for every model. [S03S04](#)

On 27 February 2024, the new owners announced wider access to maintenance instructions and technical drawings for generations of bicycles through the VanMoof Archives, explaining that some had previously been restricted to authorised workshops. Those documents cover mechanical and procedural repairs. Their release does not imply that every electronics diagram, deployment key or software right became available. [S05S06](#)

In 2026, the bicycle maker announced a connectivity change affecting older models that depended on the 2G network, with 1 April set as the subscription deadline for retaining theft tracking for specified categories. It stated that locking, riding and other independent functions **would not stop**. This event is distinct from the 2023 bankruptcy: what ages here is communications infrastructure outside the device, even after the brand and commercial activity return. [S07](#)

This sequence offers a conceptual test of three distinct causes: the loss of a commercial entity, restricted maintenance knowledge, and the ageing of an external protocol. They should not be merged into the word “failure”, or the consequence of one assigned to another. Wider release of the manuals can also be treated as a **partial counter-case** to the assumption that every supplier will obstruct continuity after restructuring.

2.4 Saudi evidence: shareable assets without an assumption of buildability

The Digital Government Authority (DGA) describes the Government Cloud Software Repository (GPaaS) as a means of storing and exchanging free and open-source government code and supporting its development. On its site, the Real Estate General Authority (REGA) presents the requirements and role of open software and a government deposit platform. [S17S18](#)

This observation matters because it identifies an **actual starting point** on which rebuild tests could be based. It does not claim that the platform itself measures restoration independence or that all Saudi software is deposited there. Public or authorised testing on defined samples is required before their portability can be judged.

3. Scientific foundation: distinguishing four kinds of evidence

3.1 Digital Preservation

Software Heritage focuses on collecting public open-source code, its development history and its availability through a traceable reference. This aim differs from preserving a complete production environment, guaranteeing the survival of the compiler used by an old release, or supplying a licence that permits every commercial dependency to be used. [S15](#)

Archiving can rescue an irreplaceable asset, yet a repository image alone may exclude external packages, operational-configuration history, database state and secrets that should never be deposited in the first place. Some elements may not be lawful to distribute. Assets should be sorted into material that can be preserved and published, material that can be preserved subject to restrictions, and assets that cannot be retained or distributed outside lawful controls.

3.2 Reproducible Builds

The Reproducible Builds project defines a reproducible build as the recreation of Bit-for-Bit identical outputs when the same source, environment and instructions are available. Environmental details that affect the result include dependency versions, locale variables, timestamps and the path from which the build runs. [S11](#)

What a match establishes: under the given assumptions and settings, the same bytes can be regenerated from specified inputs. **What it does not establish:** that the software is defect-free, that the compiler is benign, that data can be imported, that deployment keys are lawful to use, or that the competent authority accepts its operation. These belong to different classes of evidence.

Rebuildability is not always measured through bitwise identity. When platform migration or an architectural change necessarily alters the bytes, **Behavioral Equivalence** is a more appropriate criterion. It requires the inputs, outputs, error bounds and tests to be defined. Behavioural equivalence should not be described as binary identity.

3.3 Provenance and SBOM

The Software Package Data Exchange (SPDX) specification defines methods for describing software assets, their relationships and their licences. Supply-chain Levels for Software Artifacts (SLSA 1.2) supplies requirements and levels for tracking source provenance and build

provenance with varying assurances. Neither automatically grants a second party the right or capability to control signing hardware, activation mechanisms or the processor. [S12S13](#)

A complete component register is valuable evidence of readiness, but it may still encounter a **Shadow Dependency** absent from the package manifest: a remote-service licence, an embedded hardware key, a supplier-held test suite or undocumented operational knowledge. These are examples of possible mechanisms, not an inventory of vulnerabilities in any particular facility.

3.4 Secure Software Update

The Update Framework (TUF) provides an approach to limiting risks arising from update-signing keys and old or fabricated versions by using signed metadata, separate key roles and expiry dates. Running new software on a piece of hardware nonetheless remains subject to that hardware's trust rules. [S14](#)

The distinction that matters here is between **possessing a valid signature** and **possessing the authority to change the party permitted to sign**. Sound update protection may reject unauthorised software even when the device owner holds its source. Bypassing that protection should not be treated as a “recovery solution”. The alternatives are permission from the rights holder, a prior design that allows lawful updates, or viable replacement hardware.

3.5 Trusting Trust

In “Reflections on Trusting Trust”, Ken Thompson showed how examining source alone may fail to explain what a binary contains when the tools that produced it are themselves in question. David A. Wheeler later developed Diverse Double-Compiling (DDC), a method for testing whether certain compilers correspond to their intended source under defined assumptions. [S16S19](#)

This is not a demand that every organisation rewrite every compiler. It shows that “the ability to build the program” can be decomposed into assurances: preservation of tools; access to their source; diversity among verification parties; compiler suitability for the platform; and availability of evidence appropriate to the risk.

4. Conceptual model and notation

Let **s** denote a defined service and **f** an acceptable function under a supplier-loss scenario **z**. Its production path contains primitive assets and derived stages in an Artifact Dependency Graph (ADG). Some assets may be required, while others are optional alternatives. A dependency relationship should not be treated as established without inspecting the actual product.

Core variables:

- **A(v,z)**: the lawful, independent availability of a primitive asset, right or capability **v** under scenario **z**, taking the value zero or one in the simplified deterministic model.
- **Anc(r)**: the set of primitive requirements on which a reproduction route **r** depends after tracing its dependencies to their roots.
- **V(r)**: an independent judgement that the output meets the required safety and behavioural threshold, rather than merely that files compiled successfully.
- **L(r)**: an independent judgement that the route may lawfully be run and deployed, and its assets retained, under the prevailing agreement.
- **F(r)**: the set of functions demonstrated in practice by the route, rather than promised in its plan.

For each route, we propose the following measure of a necessary build condition:

$$P_r = \prod_{v \in \text{Anc}(r)} A(v, z)$$

Here, **P_r = 1** means that every primitive requirement is available in the scenario. It does not mean that the program is necessarily successful, safe or lawful. Practical reproduction also requires **V(r)** and **L(r)** to hold and the function **f** to belong to **F(r)**. When independent alternative routes succeed, a valid path may exist even if other routes fail.

In the synthetic example, we measure the number of missing root dependencies as follows:

$$B_r = \sum_{v \in \text{Anc}(r)} (1 - A(v, z))$$

This is a **count of barriers**, not an economic weighting. One irreplaceable signing key may be far more consequential than ten files recovered from another repository within minutes. **B_r** should not be compared across services without defining the assets and their boundaries.

If genuine time data become available, functional-restoration time could be studied as the time to the first successful safety-and-use test in an alternative environment **after the supplier's absence is declared**. Cases that do not succeed before the study ends would be treated as Right-Censored. We do not propose a national percentage based on the number of preserved files; it would mix heterogeneous quantities and conceal a single “blocking barrier”.

5. First executed computational experiment: repeated build from known inputs

5.1 Design

We created `analysis/demo_service.py`, containing a small Python function that checks for a synthetic identifier in a local set and returns a request identifier, a Boolean flag and a schema version. The function does not decide any real entitlement and connects to no external data.

`analysis/reproduce.py` uses the standard `zipfile` library to package three fixed elements: the service code, a small JSON schema and a file recording the reference build-time value. We sorted the filenames and imposed a uniform timestamp and consistent metadata, then built the package in two separate temporary paths. We compared the two files directly at byte level and compared their SHA-256 digests. We fetched no dependency from the internet and ran no commercial signing chain.

5.2 Actual outputs

- Number of spatially independent attempts within a single process: **2**.
- Byte-level match: **True**.
- Digest of each: `36c763c61b7114cffe473a230d5d583ca9acce14ff60bdebfab3d78747fdc52`.
- Small positive and negative service tests: **both passed**.
- The temporary result archive is not a production package and was not published as an installable update; the experiment recorded its size, the two build digests and the results of both tests in a verification file named `data/toy_rebuild_verification.json`.

The permitted scientific conclusion is narrow: with inputs and packaging information fixed in this example, the two builds produced the same result. **We do not call this an independent reproduction verified by a third party**: both attempts used the same code and the standard language interpreter in the same environment. The experiment does not establish the reproducibility of programs that use heterogeneous compilers, licences, libraries and hardware.

5.3 Failure control not executed on a real system

The experiment's limits indicate the need for later stages: running the build in two genuinely independent environments; checking asset integrity before receipt; comparing output behaviour in a test environment; testing an unavailable package repository; and testing an authorised update. These are **proposed designs**, not claimed results.

6. Second experiment: completion of the requirements graph after supplier loss

6.1 Synthetic inputs

We created `data/reproduction_graph.json`, which describes **26 nodes**. They include root assets such as service source, exported data and its schema, a local build tool and a package mirror; supplier-dependent requirements such as a proprietary compiler, development kit, activation, signing and firmware; and resulting nodes that can be derived from prior nodes.

This is an **invented educational map**. It does not represent a particular company or Saudi hardware category, and it reveals no real control points or failure modes. We separated primitive component availability from the success of a complete operational function. Each route in the experiment assumes that other tests and rights have already been satisfied, so the program neither evaluates nor attempts to bypass them.

The data define four routes:

1. `image_archive`: deployment of an archived operational image with data and configuration, plus supplier activation, signing and hardware conditions.
2. `source_escrow`: a build from available source using the original compiler and development kit, together with the original runtime requirements.
3. `source_with_local_tools`: a source build using local tools and packages; some original activation, signing and hardware requirements remain.
4. `portable_minimum`: redesign of part of the function and its assembly from assumed alternatives, with a valid licence, lawful local signing and a narrower feature set than the original.

We examined each route with the **supplier available** and then with the **supplier unavailable**. No probabilities were assigned to the two scenarios; they are deterministic condition states.

6.2 Actual outputs

Experimental route	Supplier available: number of missing roots	Supplier absent: missing roots	Completion under the second condition	Assumed functions, out of 8
Archived image	0	3	No	8, conditional
Escrowed source	0	5	No	8, conditional
Source and local tools	0	3	No	8, conditional
Portable core	0	0	Yes, under the example's conditions	6

The program ran **8 evaluations**, not eight real experiments on different products. The experiment recorded the names of missing requirements and the output for each route in a file named `data/closure_results.csv`. An internal test confirming that the number of unavailable root nodes equals the reported results passed.

The figures show that source escrow is not sufficient under a recipe that still calls for supplier assets, and that moving to a portable core may reduce dependence at the cost of certain functions. This structural result is **sensitive to how the dependencies are drawn**: if nodes or usage rights change, or alternatives emerge, the judgement changes. We did not measure repair cost, transition time or the security of the restored version.

6.3 What changes if the experiment is redesigned?

If the supplier grants an unconditional operating right and an independently valid update key before it departs, some missing nodes in the image and source routes may disappear. If the portable alternative requires source that is not licensed for modification, it fails even when the file graph appears complete. If the functional criterion requires all eight features, the portable-core route is not a success on the same terms, though it can still be described as “partial replacement”.

Precise design of the acceptance criterion is therefore not a research embellishment. It defines what we mean when we say “we restored it”.

7. The economic dimension: why not rebuild everything?

The model is not a path towards a local industry for making everything at any cost. A decision requires analysis of the Rebuild Option and the Replacement Option, including the value of keeping either open. A company or public body may pay an additional cost today for rights

and tools that reduce the difficulty of changing supplier in five years. That cost may exceed the benefit when the software has a short life and competitive ready-made alternatives exist.

We propose separating six groups of costs without inventing national figures:

1. **Cost of preserving buildability:** storing licensed source and software material, updating package mirrors, documenting versions, and running periodic tests.
2. **Cost of lost specialisation:** performance surrendered when portable alternatives replace hardware or software optimised for a particular task.
3. **Cost of knowledge:** the time of an engineer capable of understanding an old system and changing its behaviour without introducing new faults.
4. **Cost of rights and trust:** licences, approved signing methods, update maintenance and appropriate disclosure.
5. **Cost of downtime or service replacement:** the loss of a documented function, production delay and transition expenditure.
6. **Cost of complete replacement:** purchasing a new system, transforming data, training, and retiring the old system under an accepted procedure.

An organisation should compare the price of retaining its ability to modify with the price of discarding the technology altogether, not merely with the value of an old server image. Sunk Costs must be separated from avoidable future costs; returns on past investment do not justify continuing with an uneconomic product.

7.1 A scenario that weakens our thesis

If a product is available from several companies in standard data formats; if its data can be moved and tested quickly; if its function requires no deep customisation; and if it receives continuous, secure updates, retaining an independent factory to rebuild it may waste money. Practical sovereignty would then lie in **ease of replacement**, not in the organisation's ability to recreate the old program Bit-for-Bit.

8. Proposed Saudi study protocol

8.1 Design and required sample

A real-world test would require institutions to approve the selection of non-sensitive services or authorised test versions within Saudi sectors connected to four themes: government computing, economic technology services, AI components, and selected commercial digital-

infrastructure products. Functions would be classified by complexity, hardware dependence, contractual form and expected useful life rather than by supplier name alone.

A team independent of the creation team would test what it can produce from the authorised package without supplier accounts. Researchers must be denied access to real user data and operational secrets. The work should use separated laboratories and synthetic or anonymised data where needed. It requires no experiment on live critical systems.

8.2 Four recordable practical stages

Stage one: preservation of source and assets. The team inventories what it can lawfully receive in source, dependency graphs, licences, images, packages and test data. It records precisely what is missing and what is known to exist but inaccessible.

Stage two: reproduction of the build. The team attempts to create an operational version or explains why this is impossible, keeping a record of inputs, commands, tool versions and hashes. When outputs differ, it distinguishes expected differences from a failure of reproducibility.

Stage three: repair of a preselected safe problem. In a test environment, the team is asked to repair a synthetic defect or alter a narrow function against a fixed test suite established beforehand. Passing those tests must not be described as comprehensive proof of safety.

Stage four: porting a central function. The team attempts to run selected core functions on an authorised alternative platform. It records their number and acceptance results, along with differences in latency, cost and energy where reliable measurement is available. Results are calculated at service level, not collapsed into a single national index that pretends all functions are homogeneous.

8.3 What do we measure, and what do we not measure?

Proposed primary outputs: rebuild success and reason for failure; time from the start of the experiment to the first valid build; time to the first acceptable repair; number of functions that pass tests; required legal conditions; actual human effort; and the component costs of the route. Failure to complete the experiment within the observation window would be recorded as a separate status instead of assigning it a fabricated infinite restoration time.

Comparators: source alone; a binary image; source with archived build packages and runtimes; and a ready-made standards-based alternative. Their characteristics may overlap, and assignment must be specified so that the best teams do not always receive the easiest products.

Causal caveats: A successful rebuild does not establish that archiving caused success. The team may be more skilled, the component simpler, the documentation newer or the supplier previously more cooperative. Preclassification, repeated trials and a clear implementation comparison would be needed. If the data do not support a causal design, the paper should describe associations only.

8.4 Conditions for stopping or rejecting an experiment

If a test requires disrupting a production system, bypassing a manufacturer's signing key, acquiring unlicensed assets or collecting a sensitive operational map, that step must be excluded. It can be replaced with a simulation or a bounded authorised procedure. Independent reproducibility does not justify breaching the safety assurances or rights it is meant to protect.

9. Foresight to 2035: three paths for national capability

These are not calculated forecasts. They are **Conditional Scenarios** tied to observable signals.

Path one: an abundance of assets that cannot be recreated. Repositories, software images and procurement documents proliferate, but no practical test builds a version outside the supplier. Warning signs appear in contracts that preserve source without a valid environment or test tools, and in programs for which no independent team knows how to issue a new version.

Path two: indiscriminate local building without selective efficiency. Organisations try to recreate everything they purchase, including products bought more cheaply from strong international competition. Maintenance costs and delay rise, while expertise is spread across projects whose importance does not justify the effort. The decisive signal is a rising cost of independence without improvement in repair rates or ease of transition.

Path three: an economy of inheritable maintenance. Local companies and laboratories specialise in reproducing well-defined classes of products, code-migration tools, byte verification, interface restoration, behavioural-testing laboratories and the documentation of modification rights. Success is demonstrated by independent rebuilds and verifiable maintenance and export revenue, not by the number of announced plans.

9.1 Six observatory signals

1. The proportion of functions tested in practice, rather than the number of projects announcing continuity plans.
2. Rebuild success and failure rates among independent teams, classified by product type.

3. The actual duration of a supplier's support interruption and its effect on the ability to update a test version.
4. The share of products with a current package mirror and a build environment that can lawfully be rerun.
5. The difference between the actual cost of porting and the cost of buying a new replacement, accounting for reduced functionality.
6. Saudi products with customers who pay for portability, verification and repair services, rather than for published laboratory tests alone.

10. Limits of validity and generalisation

This work remains subject to six major limitations:

1. **Absence of Saudi production data:** We have no approved samples from Saudi institutional procurement, building or migration. The model therefore cannot be said to have established either fragility or strength.
2. **Simplification of the graph:** Real dependencies may contain optional conditions, cycles, version compatibility, common-mode failures and complex licensing systems. The implemented model is a simplified graph of primitive conditions with independent derived nodes.
3. **Demonstration of safety:** Availability of build prerequisites does not mean that the program is safe or produces the required results. The demonstration tests are very small.
4. **Variation in rights:** Law or contract may prohibit use of retained source, a key or firmware. We conducted no legal assessment of any particular product.
5. **The tacit-knowledge problem:** The graph does not measure the depth of engineering expertise, training time or the breadth of the error space.
6. **Difference between byte rebuilding and migration:** Moving a service to other hardware produces different bytes. It may be strategically more successful, but will not appear in a SHA-256 matching test alone.

We therefore do not call the model a “**Saudi sovereignty index**” or a “recoverability certificate”. It consists of concepts and definitions, two small experiments, and a proposed programme of real-world research.

11. Tests that could falsify the thesis

For the contribution to be scientific, we must specify what would make it less convincing:

- If independent service providers can consistently purchase and operate safe new alternatives faster and at lower cost than rebuilding originals, the general value of retaining build capability would be weakened for the product classes studied.
- If retaining the build chain adds no predictive power once team expertise, software simplicity and supplier strength are accounted for, the claim that the Reproduction Boundary is an important independent measure should be revised.
- If no “core function” can be specified without losing essential rights or features, the portable-core model would be inappropriate for that service even if it were technically executable.
- If technical evidence shows that a dependency described as exclusive can immediately be replaced by a trustworthy, licensed alternative, it must not remain a “blocking barrier” in the graph.
- If most programs have short lives and can be replaced efficiently, preserving broad engineering capability may be more valuable than preserving build recipes for products that age quickly.

This is the difference between an enterprising proposal and a closed claim: asking what result would compel us to change the strategy before billions are spent on it.

12. Reproduction protocol and accompanying research materials

Original experiment file architecture as described by the manuscript:

File	Contents	Description of its limits
analysis/demo_service.py	A synthetic service whose logic is fully disclosed	Contains no real data, customers or entitlements
analysis/reproduce.py	Test that creates the package twice and checks the dependency graph	Uses only standard Python
data/reproduction_graph.json	26 nodes, four routes and a test of two conditions	Authorised fictional inputs that do not represent Saudi reality
data/closure_results.csv	Eight detailed outputs and the cause of missing dependencies	Computational cases with no probability weights
data/toy_rebuild_verification.json	Package size, two build digests and the results of two tests	Local check, not independent multi-institution verification
analysis/README.md	Execution steps, assumptions and verification methods	Publishes no keys or operational data

The manuscript describes the original procedure as running `python3 analysis/reproduce.py` from the working path and then comparing the CSV and JSON outputs. The described procedure assumes no software dependencies beyond the standard Python library. A different ZIP or Python implementation may expose a difference in the package file in a third environment; if it does, the environment must be fully characterised before concluding that the underlying idea has failed.

Materials accompanying this edition are `closure_results.csv`, exported from the supplied evidence workbook; `source_register.csv`; and `news_casebook.md`. They are available in [Research materials](#). The research PDF is available separately.

Archives created inside the temporary directories are not published as ready-to-run programs. Only the verification result is retained, and the package is created afresh when the program runs.

13. Research conclusion

The paper's conclusion is not “we should distrust suppliers”. The supplier may be best placed to restore operation and most capable of sustaining innovation. Nor is the conclusion “open source solves everything”. Its tools may age, its expertise may disappear, and it may be unable to perform functions subject to complex rights.

The more specific finding is that **digital ownership can be separated from the engineering capability to revive a function**. An organisation can retain all the material yet lack a lawful, tested path connecting it to a function. A small organisation can retain an effective route to repair despite possessing few physical assets.

The two small experiments demonstrated the possibility of building one package twice with Bit-for-Bit identity in a known environment, and showed how losing a limited number of roots in a synthetic dependency model can break routes even when the remaining material is preserved. Whether extending this principle to Saudi services yields a real return in cost, time, independence and maintainability is a **hypothesis requiring an independent experiment**.

If that experiment is conducted, it will give us something deeper than a slogan of sovereignty: testable knowledge of what a system can make again, who can make it, on which device, and with costs, rights and safety understood by those who will pay the price if it fails.

Primary references and traceability markers

Full details, publication and event dates, source limitations and citation branches appear in `source_register.csv` and `news_casebook.md`. The corresponding identifiers are retained here:

1. [S01](#) National Aeronautics and Space Administration (NASA), *Voyager 1 Resumes Sending Engineering Updates to Earth*, 22 April 2024.
2. [S02](#) National Aeronautics and Space Administration (NASA), *Voyager 1 Resumes Sending Science Data from Two Instruments*, 22 May 2024.
3. [S03](#) VanMoof Bankruptcy Trustees, *Fourth Public Bankruptcy Report*, 6 June 2024, recording the event of 17 July 2023.
4. [S04](#) Cowboy SA, original *Bikey* app page, July 2023.
5. [S05](#) VanMoof, statement on new ownership and the recovery programme, July 2024.
6. [S06](#) VanMoof, *VanMoof Archives for Legacy Models*, 27 February 2024.
7. [S07](#) VanMoof, *What's Happening with 2G Network Shutdown*, 2026 guidance.
8. [S08](#) Fisker Owners Association, owners' association statement filed in the company's US bankruptcy case, 26 July 2024, p. 2. The original filing was indexed, but the file server did not respond to a direct request during final verification.
9. [S09](#) Undercurrent Inc., developer/provider of *OceanLink Pro*; the App Store description identifies it as the official app for FOA members.
10. [S10](#) Reuters, report on approval of Fisker's bankruptcy plan, 11 October 2024.
11. [S11](#) Reproducible Builds, *Definitions*, technical definition of identical builds.
12. [S12](#) SLSA, *SLSA Specification Version 1.2*, approved on 24 November 2025 and current at the evidence cut-off.
13. [S13](#) SPDX, *Specifications*, covering software components and their licences.
14. [S14](#) The Update Framework, *Overview*, secure software updates.
15. [S15](#) Software Heritage, *FAQ*, long-term preservation goals.
16. [S16](#) Ken Thompson, *Reflections on Trusting Trust*, 1984.
17. [S17](#) Digital Government Authority, *Government Cloud Software Repository (GPaaS)*.
18. [S18](#) Real Estate General Authority, *Free and Open-Source Government Software*, undated official institutional page.
19. [S19](#) David A. Wheeler, *Fully Countering Trusting Trust through Diverse Double-Compiling*, 2010.

Note on quotations: The text uses attributed summaries of published ideas. It attributes no fabricated dialogue to an engineer, rider or researcher. No statement by an interested party or app-store page is presented as an independent technical review of all functions.