Digital Twin Life Cycle Considerations

1. Learning objectives, scope and prerequisites

1.1 Learning objectives

By the end of this chapter you will be able to:

1. **Distinguish** the asset life cycle from the twin life cycle, and **identify** the coupling points at which an event in one obliges engineering work in the other.
2. **Derive** a revalidation policy for a twin already in service: choose a drift signal, set a threshold from the twin's stated purpose, and build an action ladder with a named owner on every rung.
3. **Design** a co-versioning scheme that keeps an asset configuration, a twin release and a telemetry schema mutually compatible, and **predict** the failure that results when one of the three moves alone.
4. **Specify** a retention and replay policy for a twin's data, and **compute** what it costs in storage, in reprocessing, and in questions the twin can no longer answer.
5. **Plan** the retirement of a twin: what is archived, what is destroyed, what is handed over, and which obligations outlive the asset itself.

Every exercise in Section 10 names the objective it exercises. An exercise that mapped to none was cut.

1.2 What this chapter covers

This chapter is about what happens to a digital twin *after* the lab prototype. It treats the twin as a long-lived production system with an unusual property: a second system, made of steel and silicon and owned by somebody else, is changing underneath it, and neither system can be frozen while the other catches up.

Five phases are treated in order – conception, construction and commissioning, operation, evolution, retirement – then four concerns cutting across all five: co-versioning, data retention and replay, interface evolution, and deployment topology with operational ownership. One case study, a fleet of grid-connected battery energy storage sites, supplies three worked examples with progressively less support.

1.3 What this chapter deliberately does not cover

How to build a twin. No modelling technique, no co-simulation orchestration, no sensor selection, no platform comparison. You are assumed to have a twin already, or to be inheriting one.

The mathematics of model updating. Bayesian calibration, sequential filtering and discrepancy modelling are named where they are the mechanism, and pointed at in Section 11; none is derived. The concern here is *when* an update should be triggered and what it costs an organisation, not how the estimator works.

Security engineering. A twin enlarges an asset's attack surface, and that changes two life-cycle decisions here – what a retention tier may hold, and what a retirement must destroy. Both appear where they bite.

Regulation as law. Retention obligations are treated as an input the engineer receives, expressed as a horizon and a set of questions that must remain answerable. No jurisdiction is named as authoritative.

1.4 What you are assumed to know

You are an experienced software engineer. This chapter does not re-teach what a twin is, or what goes into building one. You are fluent in distributed systems, release engineering, data retention and interface design, so the chapter uses – without explanation – semantic versioning, schema migration, canary release, storage tiering, idempotency, event sourcing, and the distinction between a stream and a batch.

What you may not have thought about is year seven: the asset has been re-instrumented twice, the telemetry schema has moved three times, the engineer who calibrated the model has left, and somebody is asking the twin a question whose answer will be read out in a contractual dispute. That is the subject.

1.5 Terms this chapter pins down

Three pairs of words are used loosely in industry and precisely here.

Fidelity and fitness for purpose. *Fidelity* is how closely the twin's output matches the asset's measured behaviour, on a named quantity, against a named tolerance. *Fitness for purpose* is whether that fidelity suffices for the specific decision the twin was commissioned to support. A twin is never "accurate" in the abstract. It is fit for a stated purpose or it is not, and fitness for purpose is what a twin loses over time.

Recalibration and revalidation. *Recalibration* re-fits a model's parameters to recent data: cheap, routine, often automatic. *Revalidation* re-establishes that the twin is still fit for its stated purpose and produces a dated, signed result: expensive, occasional, human. Conflating them is how a twin comes to be trusted for years on the strength of a claim nobody has re-examined.

Decommissioning and retirement. *Decommissioning* takes the twin out of live service: disconnecting it and stopping synchronisation. *Retirement* is the whole end of life, of which decommissioning is the first step and archival the last. Teams that plan the first and not the second discover, two years later, that the data they need is in a storage account nobody is paying for.

- An **asset stage** is where the physical thing is — conceived, designed, constructed, commissioned, operating, refurbished, decommissioned. It changes on the timescale of years.
- A **runtime phase** is where the running twin process is — unbound, bound, synchronised, out of sync, done, stopped. It changes on the timescale of seconds.
- A **release cycle** is the twin software's own build-and-ship loop. It changes on the timescale of sprints.

The literature does not agree on these words, which is precisely why this chapter fixes them. Kamburjan et al. write *stage* for the asset side [1]; Picone et al. write *phase* for the runtime side and take care to distinguish a phase from a *state* [2]; Michael et al. use *life cycle phase* for the asset and a DevOps continuum for the software [3]. Where this chapter cites one of them it names their term once and then returns to its own.

2. Why the life cycle is where twins fail

2.1 The characteristic failure is silent

A twin that is wrong on the day it ships gets fixed. A twin that becomes wrong slowly, over three years, while continuing to return plausible-looking numbers, does not – because nothing about it looks broken. No exception, no alert, no failed health check. The dashboard is green, and the predictions have stopped meaning what the people reading them believe they mean. This is a life-cycle failure rather than a modelling one: the model was fit for purpose when it was validated, then the asset or the instrumentation or the operating regime changed, and no process existed that would notice.

Michael et al. name this directly as *time complexity*: the long lifespan of the actual system against a changing reality, sitting alongside system, integration and information complexity as the underlying difficulties in twin engineering [4]. Their observation that most twin practice is still "artisanal" is the same problem restated — an artisanal system is one whose knowledge lives in the people who built it, and twenty-five years guarantees those people leave.

The literature reaches the same place from a different direction. Analyses of twins for intelligent systems observe that a twin's predictions go stale wherever the world it represents moves quickly, and treat the twin's knowledge as something that can drift, decay, or fall behind what the asset is now doing [1]. Knowledge as a perishable good with a shelf life is the right framing for an engineer: it converts a vague worry into a maintenance obligation with a period.

2.2 Twins are funded as projects and consumed as services

The economics push the wrong way. A twin is usually funded as a project, with a business case, a delivery date and a demonstration – and that funding model ends at the demonstration. What follows is a service with a fifteen-year obligation and no budget line.

Surveys describe the consequence as a gap between artisanal and industrial practice: twins are built with great care by small expert teams, and the discipline of sustaining them at scale is what remains missing [2]. That scaling problem is usually read as one of *quantity*. It is at least as much one of *duration*: a single twin sustained for fifteen years exercises the same weaknesses – nobody owns it, nothing revalidates it, its dependencies rot. Interview studies of continuous integration and delivery in cyber-physical systems find the same asymmetry inside organisations otherwise competent at software delivery, with validation against real hardware the part that resists automation [3].

2.3 The asset outlives several generations of your stack

A grid-connected battery storage site has a design life of fifteen years. An offshore structure has thirty or more, and the live engineering question about many of them is life extension past the original design life [5]. Over fifteen years you will change container runtime, message broker, time-series store, cloud provider, language runtime, and the simulation tool that produced the plant model. Several of those changes will be forced on you. The asset changes too, on its own schedule: a component replaced with a non-identical part, a control system upgraded, a sensor moved. The twin must survive both sequences, and the two are not coordinated.

2.4 The value is realised late

The expensive question a twin answers is rarely asked in year one. In year one you ask "is the plant running normally?", and a dashboard over live telemetry answers it. In year eight you ask "has this asset been operated inside its warranty envelope for eight years, and can you show me?" – and only a system designed in year one to still answer that in year eight can do so.

A comprehensive review makes this concrete for batteries: the payoff case it constructs is choosing the moment a cell should *leave its first application and move to a second life* – a decision taken at the end of service life and informed by everything that preceded it [6]. That decision is worth more than every dashboard the twin ever rendered, and it is unavailable to a twin whose history has been downsampled away. Life-cycle work is therefore not overhead protecting a delivered value; it is the mechanism by which most of the value arrives.

2.5 The running case study

One case study carries the chapter. It is constructed for teaching, not lifted from a paper, and it is deliberately mundane.

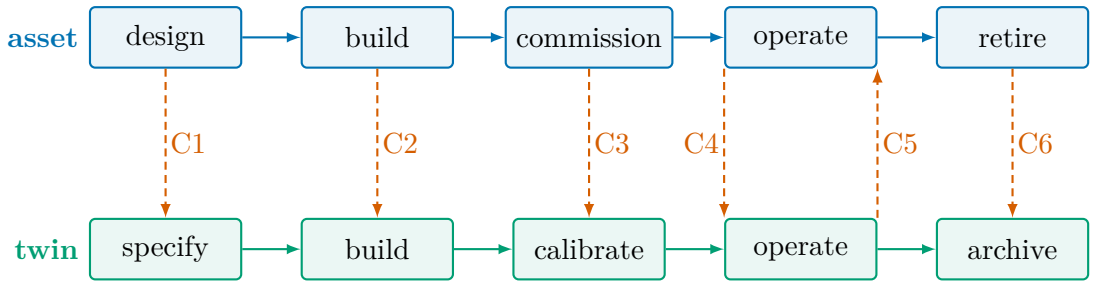
An operator runs **twelve grid-connected battery energy storage sites**. Each is rated 20 MW / 40 MWh and built from 1,920 modules in 24 containers. The sites were commissioned between 2023 and 2025; design life is fifteen years. Two contracts constrain the operator:

- a **capacity warranty** from the cell supplier, under which the operator must demonstrate annually that each site's measured state of health – the ratio of present usable capacity to rated capacity – is above a schedule, and must show the site was operated inside a stated temperature and depth-of-discharge envelope;
- a **grid-services contract**, with availability and response-time obligations and financial penalties.

A twin exists per site, plus a fleet-level twin aggregating them. Each site twin was commissioned to support exactly three decisions:

Table 1: The purpose table for a site twin: three decisions, each with a named quantity and a stated tolerance.

	Decision	Fidelity tolerance
P1	Annual state-of-health declaration for the warranty schedule	absolute error at most 1.5 percentage points of rated capacity
P2	Day-ahead sizing of the dispatch bid	predicted energy throughput within 3% over a 24-hour horizon
P3	Pre-flight check of the thermal envelope before an unusual dispatch profile	peak module temperature within 2 K, conservative bias required



C1 purpose and fidelity tolerances fixed C2 as-built record handed over
C3 first signed fidelity claim C4 asset modified, twin release follows
C5 drift threshold crossed, revalidation triggered C6 archival record created

Figure 1: The asset clock and the twin clock, with six coupling points. Between coupling points the two clocks run independently, which is what makes them tractable; at a coupling point they do not.

Keep the three rows of Table 1 visible. Every row pairs a decision with a tolerance, and it is the pairing that does the work: no row asserts that the twin is accurate, only that it is accurate enough for one named use. Almost every judgment in the rest of the chapter is made by asking which of P1, P2 or P3 a proposed change would break. A twin with no such table cannot answer any life-cycle question except by argument.

3. Two clocks

3.1 The model

A twin project runs on **two clocks**.

The **asset clock** is the physical asset's progression: conception, design, construction, commissioning, operation, modification, retirement. Owned by the asset owner and driven by capital budgets, regulation, weather and physical failure, it is slow and not negotiable with you.

The **twin clock** is the twin's progression as a software system: specification, build, calibration and commissioning, operation, evolution, decommissioning. Owned by the twin engineering team and driven by releases, dependency upgrades, drift and incidents, it is fast.

Between them, at specific moments, are **coupling points**: places where an event on one clock obliges work on the other. Figure 1 sets the two clocks side by side; read down the arrows, and note that they are sparse. The long stretches without one are where the two teams can work without consulting each other, which is the property the whole model is for.

3.2 Why two clocks and not one process

The obvious alternative models the twin as a work product inside the asset's own systems-engineering process – a real and well-developed position. Work applying model-based systems engineering to twins proposes a V-model spanning the twin's entire life cycle from

conception to decommissioning [7], and treatments of twin engineering processes map the technical processes of the systems-engineering standard onto twin-enabled systems, including retirement, which may mean transfer to a new life cycle, disassembly, repurposing or recycling [8]. That literature is where the vocabulary of phases comes from.

The reason this chapter keeps two clocks is practical. **The two life cycles have different owners, different cadences and different change-control regimes.** A twin release can ship on a Tuesday. A change to the asset requires an outage, a permit, and a window agreed six weeks ahead. Modelling them as one process throttles the fast clock to the slow one, which is how twins end up frozen for eighteen months awaiting an asset change that then arrives in an unplanned form. Two clocks with explicit coupling points is the trade a distributed systems engineer makes routinely.

3.3 The six coupling points

C1 – purpose is fixed. Somebody decides what the twin is for. This is when the P1/P2/P3 table gets written and fidelity tolerances become numbers rather than adjectives. Without a stated purpose there is no way to decide whether a drift of 0.4 percentage points matters. Fitness for purpose as the organising question, rather than accuracy in the abstract, is the position taken by the major research-gaps assessment of the field, which ties it directly to uncertainty quantification [9].

C2 – the as-built record is handed over. The built asset is not identical to its design: cable runs differ, a supplier substituted a part. The as-built record is the twin's ground truth, and handover is when the twin's parameters stop being assumptions. It is the most commonly botched coupling point, because it falls in the asset project's busiest phase and its recipient is a team the construction contractor has no obligation to.

C3 – the first fidelity claim is signed. The asset is commissioned, the twin calibrated against real measurements, and somebody signs a statement that the twin meets its tolerances. Note *signed*: a fidelity claim with no date and no author is a memory, and memories are not revalidated because nobody knows they exist.

Virtual commissioning is the well-known payoff here: in process industries, twins complete commissioning work before the facility is ready, compressing a phase that carries real safety risk [10]. It is also a trap. A twin used *for* commissioning was validated against a design, not against the built asset – different claims, and the second still has to be made.

C4 – the asset is modified. A twin release must follow, and until it does the twin is answering questions about a system that no longer exists. Section 5 is about this coupling point.

C5 – drift crosses a threshold. The only coupling point that fires from the twin side. Worked Example 1 designs it.

C6 – retirement. The asset is retired, repowered or sold; the twin is decommissioned and an archival record created. Section 4.5 is about the fact that this has deliverables.

3.4 What the model buys you

Three otherwise-vague questions become answerable. *Whose work is it?* At a coupling point, work is owed by a named party; away from one, each clock is free – and most twin

disputes are really disagreements about whether a coupling point has fired. *When may we move fast?* Between coupling points, at software cadence. *What must be recorded?* Exactly the coupling points: a purpose table, an as-built handover, a signed fidelity claim, a release note tied to an asset change, a revalidation result, an archival record. That set is the twin's audit trail, and it is small enough to maintain.

4. The five phases

4.1 Conception: bind the twin to decisions, not to the asset

Every twin is built for a purpose, and the purpose determines the models, which determine the telemetry, which determines the retention bill for the next twenty years. Lugaresi and Vangheluwe make the dependency explicit: the selection of goals and properties of interest drives every downstream choice, illustrated with a resistor — if the property of interest is geometry you need a computer-aided design model. Two legitimate twins of the same object can share nothing.

The long-lived part: Michael et al. observe that because twin lifetimes are long, the questions users ask will change, the purpose drifts slowly, services must evolve with it, and the underlying models may stop being accurate or complete enough. They also name the tempting escape and its price — collect and integrate as much data as possible against future purposes, which buys flexibility if the change can be predicted and makes the models harder to use for the purpose you have today [4].

The failure at conception is a twin specified as a *representation* rather than as an answering machine. "A digital twin of the storage site" has no acceptance criterion, no bound on scope, and no basis for declining a feature request. Specify instead a small set of decisions the twin must support, each with a tolerance – the P1/P2/P3 table. This is the discipline of writing service-level objectives before writing a service, and it pays off the same way: arguments become measurements. Three things follow.

Scope has a boundary. A model detail earns its place if some decision in the table needs it. The battery twin does not model the site's lighting circuits, because no row requires it. Without the table this is a matter of taste, and taste loses to enthusiasm.

Fidelity becomes plural. The three rows have different tolerances on different quantities, and one model will not sit at the same operating point for all three. P3 requires a *conservative bias*: for a safety envelope check, a model that over-predicts temperature is useful and an unbiased one is not. Governance frameworks make the related point that confidence in a twin's outputs must be stated alongside the outputs, which requires the sources of uncertainty to have been identified first [11].

The retirement criterion exists from day one. If the twin is for P1, P2 and P3, its end of life is when no decision in the table is still being taken – testable, and the only defence against a twin that outlives its usefulness while consuming an operations budget.

Purpose tables must be *renegotiable and versioned*. A twin will be asked for a fourth decision, which is good: it means the twin is trusted. The failure is answering it without adding a row, because the new answer then inherits a fidelity claim never made for it.

4.2 Construction and commissioning: ground truth arrives once

Between C2 and C3 the twin acquires the two things it can never reconstruct later.

Commissioning is where a twin acquires the two things it cannot acquire later: an identity mapping, and a baseline.

Identity. Picone et al. describe binding as the step where the twin associates itself with a physical twin whose functionality and properties are the ones it was built to expect, and which it can actually talk to [2]. In a lab that is a configuration line. On a real site it is a data-quality problem with a long tail: the tag in the supervisory control system, the serial number on the nameplate, the asset number in the maintenance system and the identifier in the twin registry are four strings maintained by four teams, and at least one is wrong on any given day.

The rule that survives twenty years: **binding must be verified, not asserted.** A twin that is told which asset it is looking at will eventually be told wrong. A twin that checks — does the observed usable capacity match the nameplate rating, does the serial number on the container's fieldbus match the registry — refuses to bind instead of confidently misreporting. Refusing to bind is a visible failure with an owner. Binding to the wrong asset is discovered by a rejected warranty claim. §9.1 develops the registry side.

Baseline. The second irreplaceable artefact is the record of what the asset did when it was new and healthy. Every later drift computation is relative to something, and if you did not capture that something during commissioning you will spend the twin's life comparing against a period that already contained the fault you are hunting.

Capture at commissioning, and store as a first-class immutable artefact:

- The acceptance test data at the sample rates used during the test, not the reduced rates used in normal operation.
- The models as they stood, with their parameter values and the fitting data.
- The stated validity envelope: which flows, temperatures, speeds and valve positions were actually exercised.
- The identity mapping, with the evidence used to verify it.

That bundle is the twin's birth certificate. It is small, it is written once, and in year nine it is worth more than any dashboard.

The industrial framing for this chain is the *digital thread*: the traceable link from as-designed and as-built records through to what consumes them. Standardisation work reports it as a practitioner requirement rather than a research aspiration, with a surveyed expert stating that twin versioning is a prerequisite for managing correct information about the twin in the context of the digital thread [12]. Note the direction: versioning is not layered on top of traceability, it is what traceability is made of.

Commissioning calibration is the cleanest data you will ever get. The asset is new, undamaged, fully instrumented, and available for tests unthinkable during revenue operation – for the storage fleet, the only period in which a full characterisation cycle on every container is affordable. That baseline bounds every later drift measurement. Two practices repay their cost:

- **Record the calibration inputs, not only the fitted parameters.** A fitted pa-

rameter set cannot be re-derived under a different method. The raw characterisation data can.

- **Sign the fidelity claim per purpose row.** "The twin meets P1 at 0.4 percentage points, P2 at 1.9%, P3 at 1.1 K conservative, measured on 2025-03-14, by X, using data Y." This artefact is what a revalidation three years later compares against.

4.3 Operation: drift, and the cost of being believed

Operation is the phase in which the twin is *believed*, and belief is what makes drift dangerous. Two distinct degradations need separate names.

Twin drift is divergence between the twin's predictions and the asset's measured behaviour, on a named quantity, over time. Its causes are physical – the asset ages, components degrade, the regime shifts – and its remedy is recalibration or revalidation. For the storage fleet the dominant driver is capacity fade: hybrid approaches coupling an empirical fade model with machine learning report meaningfully different error inside and outside their training distribution [13]. The engineering lesson is not which estimator to pick. It is that the out-of-distribution case is where a twin is both most needed and least trustworthy, which argues for measuring drift continuously rather than assuming a validated model stays valid.

Configuration drift and schema drift are divergence between what the twin believes about the asset's configuration or telemetry format and what is true. Their causes are administrative, their onset is sudden, and recalibration will not fix them – it will absorb them, fitting parameters to compensate for a units error and producing a model that looks healthy and is wrong. This is the most dangerous interaction in the chapter: **an automatic recalibration loop will silently launder a schema error into a plausible model.** Section 5 handles it by making schema changes detectable, not by making the recalibration cleverer.

A third concern is peculiar to twins. **Time discrepancy**: the twin and the asset do not share a clock, and the twin's computation takes time. Whether an answer is about *now* or about a slightly stale past is a design parameter, studied directly as the discrepancy between digital and physical twins [14]. P3 is where this bites: an answer arriving after the dispatch has started is not a late answer, it is no answer. Attach a validity horizon to each purpose row and treat exceeding it as a failure, not as latency.

- **Model drift.** The asset is where it was but the model's agreement has degraded, because a fitted parameter has gone stale or a learned component was trained on a distribution that no longer holds. Repair: recalibrate or retrain, then revalidate.
- **Representation drift.** The schema, units, tag mapping or component structure no longer describes the asset. Repair: change the twin's structure — Kamburjan et al.'s architectural self-adaptation [15] — and revalidate.

The twin's *services* also age differently. What-if services examine alternative scenarios in parallel, including design-space exploration and deliberate fault injection, feeding diagnosis, predictive maintenance and reconfiguration [16]. Monitoring degrades gracefully when the twin drifts – the comparison gets noisier but keeps meaning. What-if degrades treacherously, because a counterfactual has nothing to be compared against: a drifted twin's what-if answer is unfalsifiable at the moment it is given.

4.4 Evolution: the asset changes, and so does everything else

Evolution is the long middle, with three independent sources of change: the asset, the twin's own stack, and the questions.

Asset change is the coupling point with teeth. Some asset changes are parametric – a component replaced with an identical one – and the twin absorbs them by updating a value. Others are structural: a container removed, a different inverter fitted, the site re-zoned. A structural change invalidates the twin's *architecture*, not only its parameters.

Life-cycle management work draws the distinction explicitly, separating *behavioural* self-adaptation – adjusting an existing controller to keep maintaining a goal – from *architectural* self-adaptation, reconfiguring the twin's own structure so that components of the physical system remain correctly matched by the corresponding twin components as the asset moves between life-cycle stages [15]. Read as a software engineer: the first is a parameter change, the second a topology change, and a system that can only do the first will quietly mis-attribute measurements after a structural asset change.

The reverse direction matters too. A twin can drive reconfiguration of the asset, and supporting that with a degree of autonomy is part of what distinguishes a twin from a passive digital shadow [17]; reconfiguration-management frameworks treat selecting and planning a new configuration as a problem in its own right [18]. The consequence is uncomfortable: if the twin can change the asset, the twin's release process is part of the asset's change-control regime, and a twin deployment is a plant modification.

Stack change is ordinary software rot with one twist. A twin's dependencies include *simulation tools*, whose obsolescence profile differs from a library's: a plant model may be authored in a tool with a licence, a vendor and a file format that outlives neither. The field's mitigation is model exchange through a standardised interface – the Functional Mock-up Interface packages a dynamic model with its interface description so it can be exchanged between tools, and is one of the fixed points co-simulation practice is built on [19]. Treat a proprietary, non-exportable model as a dependency with a support horizon, and record that horizon.

Keeping models current is itself an operational activity. Dittler et al. address exactly this for modular production systems: simulation models built in the development phase get used in the operational phase for prognosis and operation-parallel simulation, and the operator then faces the problem of adapting a heterogeneous landscape of models to maintain application-oriented realism when the asset, its environment, or the set of applications changes [20]. Their answer is continuous model adaptation organised as a plan-do-check-act loop with agents. Whether or not you want agents, the framing is the one to steal: **model maintenance is a standing operational workload with a cadence, not a project that finishes.** Staff it, or it will be done by whoever has the incident.

Co-evolution is the standing requirement, and the transition period is where it bites. Evolution is where the release cycle and the asset stage grind against each other, and it is the stage most likely to kill a twin quietly.

Michael et al. state the requirement: the relationship between the two life cycles demands constant co-evolution, going beyond bidirectional data updates to include updates to the *structure* of the models the twin keeps. They then raise the questions a practitioner

actually faces — how and when a twin should be disconnected from and reconnected to the actual system, how representations get updated, and how model, metamodel and data consistency is maintained *during the transition period* [4]. That last clause is the whole engineering problem: the transition is not instantaneous and the asset does not pause.

The revalidation debt. Every change — to the asset, a model, the schema, the telemetry pipeline — either preserves the existing acceptance evidence or invalidates part of it. A change that invalidates evidence and ships anyway creates a *revalidation debt*: a dated, named claim the twin no longer has support for. Track it with three fields — which claim, which change invalidated it, what evidence would discharge it.

Make it explicit because it is otherwise invisible. A twin with six months of accumulated debt looks identical on every dashboard to one with none, and fails identically too, right up until somebody asks it a question in the region where the evidence went missing.

Learned components make evolution a data problem. Lu et al. address a cyber-cyber twin of Norway's cancer registry system, which evolves constantly as diagnostics and technology change; keeping the twin synchronised needs abundant data labelled by the *new* version of the real system, so their method treats the previous twin's model as a pretrained starting point and fine-tunes it on a subset of cases selected to be maximally informative [21]. Xu et al. apply the same instinct in a cyber-physical setting [22]. The transferable lesson needs neither technique: **the expensive part of evolving a learned component is acquiring labels from the system as it now is**, so plan a labelling budget and a selection policy in advance. A team that plans only the retraining meets the labelling cost at the worst moment.

Structural evolution differs in kind from parametric evolution. Kamburjan et al.'s separation is the one to hold on to: adjusting an existing controller is behavioural adaptation, whereas adding or replacing a component is architectural adaptation, and these want different machinery [15]. In release terms, a parametric change ships as a minor version with a recalibration record; a structural change is a major version with a migration. Pretending otherwise is how a twin ends up with two incompatible notions of what a container is.

4.5 Retirement: a phase with deliverables

Retirement is where twin practice is thinnest, and the gap is structural: the people who commissioned the twin have moved on, and nobody's objectives include shutting it down well. Distinguish three endings, because their deliverables differ.

Decommissioning the asset and decommissioning the twin are separate decisions with separate dates, and the twin's date is later — sometimes by a decade. Put both in the decommissioning plan, in writing, with an owner for the gap.

Four reasons a twin outlives its asset, each setting a different retention period:

1. **Evidence.** Warranty, insurance, incident investigation and regulatory reporting all reach backwards into operating history. The period is set by the longest applicable limitation period — a legal input, not an engineering one.
2. **Fleet learning.** Site 3 is one of twelve. Its fifteen years of degradation history is a prior for the next site's models.

3. **The replacement asset.** A twin of the repowered site that begins with no history cannot distinguish a new fault from a longstanding one.
4. **Life-cycle accounting.** Environmental and cost reporting is computed over an asset's whole life, including disposal, so it can only be completed after the end.

The asset is retired and the twin with it. What survives is the archival record: the frozen bundle that can still answer questions after every live system is gone. Early conceptual work anticipated this by including a *bill of disposal* among the information sets a twin instance carries through the life of the product it is linked to [23] – a striking thing to have specified at the outset, and still routinely absent from implementations.

The asset is modified so heavily that the twin is replaced. Repowering, refit or control-system replacement. The interesting question is what the successor inherits: the calibration history, certainly; the model, probably not. Life extension work treats this as a normal event in an ageing asset's life [24].

The asset is sold or transferred. The twin, or a redacted version, goes with it, and every unstated assumption about data ownership surfaces. Governance treatments name accountability, responsibility and ownership of the intellectual property embodied in a twin as things a framework must settle explicitly [11]. Unsettled by the time of sale, it gets settled expensively.

Two obligations survive in every case.

Retention obligations survive. The warranty schedule runs to year fifteen whether or not the twin does. Section 6 is about designing so that this remains possible.

Security obligations survive. A retired twin is a detailed model of a real facility, often including its control logic and its weaknesses. Security analyses observe that the digital representation contains critical information letting an attacker map the system or derive private patterns from its databases, states and configurations [25]. A decommissioning plan leaving that material in an unowned storage account has created a liability nobody is monitoring. What is archived must be archived deliberately, what is not must be destroyed, and both decisions written down.

5. Cross-cutting concern I: co-versioning and release engineering

5.1 The compatibility triple

A twin's answer is a function of three things that change independently:

- the **asset configuration** – what is physically installed, as maintained;
- the **twin release** – the versioned bundle of models, code, configuration and calibration;
- the **telemetry schema** – the format, units and semantics of the data arriving from the asset.

Call these the **compatibility triple**. An answer is trustworthy only when all three agree. Follow the three lanes across Figure 2: what matters is not that they move, but that they move at different moments, so that the intervals in which they disagree are the intervals in which the twin must not be believed.

The lesson is one a release engineer will find familiar and an asset engineer will not: **the**

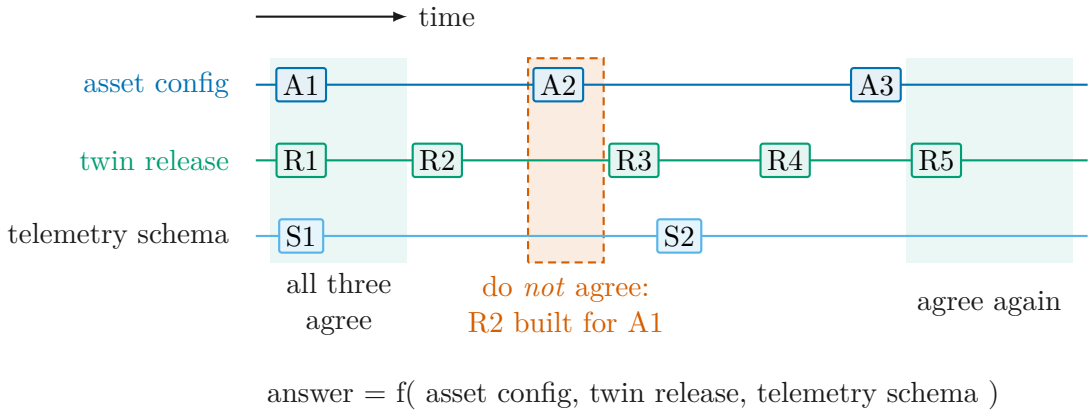


Figure 2: The compatibility triple. The three lanes move on different clocks and for different reasons. The engineering task is not to keep them in step – that is unachievable – but to make disagreement detectable and to refuse to answer while it lasts.

goal is not synchronisation, it is detectable divergence. You cannot make an asset change and a twin release atomic. You can make it impossible for the twin to answer as though they were.

5.2 What "versioning a twin" actually means

A twin release is not one artefact, and that is what makes twin versioning harder than application versioning. One release pins: executable code; models, which may be binaries from a tool you do not control; calibration parameters; configuration describing the asset's structure; the schema the telemetry must satisfy; and the identity of the specific asset instance. Change any one and the answers change.

Standardisation work records twin versioning as a practitioner requirement, motivated by traceability across the asset's life from creation to end of life [12]. Approaches to automating twin life-cycle management go further, combining a declarative twin description language with a pull-based, version-controlled deployment framework, so that core operations such as connecting and synchronising become things the platform performs rather than things an operator does by hand [26]. The underlying pattern – declarative desired state in a repository, with reconciliation – is standard software operations practice [27]. Two adaptations are needed before it transfers.

Rollback is asymmetric. Rolling back a twin release is routine. Rolling back the asset is not. A twin release touching a control or reconfiguration path therefore has no rollback in the sense you are used to, and needs the corresponding safeguards: staged rollout across the fleet, an abort criterion defined before the rollout, and a manual gate.

A twin's canary is a fleet, not a traffic slice. You cannot route 5% of a site's telemetry to a new release and compare user metrics. You can deploy it to one site, run it in shadow beside the current release, and compare outputs on identical inputs for a defined period. That is why the fleet-level twin earns its cost.

5.3 Where continuous delivery stops

There is a documented gap between what twin teams say they want and what they will automate. The standardisation survey that records twin versioning as a requirement also reports that 81% of respondents rated continuous deployment as *not* important [12]. Read that carefully rather than as conservatism: continuous deployment means a change reaching production without a human decision, and when production is a plant that differs from a web service. The interview evidence is consistent – pipelines exist and are valued, but validation against real hardware is where automation stops [3].

The productive move is to separate *continuous integration* from *continuous deployment*. Everything up to the gate – building the release, running models against recorded telemetry, checking that outputs match the previous release within tolerance, verifying the schema contract – is automatable, valuable, and mostly missing in practice. The gate itself can stay human. Work aimed at this territory connects design-time models to runtime monitoring so the operations loop closes by construction [28], packages that connection as an explicit process for building twins from engineering artefacts [29], and applies it to the systematic evolution of twins for built assets [30].

5.4 A practical scheme

- **Every stored answer carries the whole triple.** Not a timestamp: the asset configuration identifier, the twin release version and the schema version, written beside every persisted output. This is the highest-value practice in the chapter and it costs three columns.
- **The twin refuses rather than extrapolates.** A release declares the asset configurations and schema versions it supports; presented with anything else it returns a refusal, not a number. An engineer's instinct is to degrade gracefully. Here graceful degradation is the failure, because a plausible number gets consumed as a real one.
- **Asset change proposals are reviewed by the twin team.** Not to veto them, but to classify each as parametric or structural and schedule the release. This makes C4 a process step rather than a surprise.
- **The purpose table is versioned with the twin.** A release changing a tolerance is a major version, because it changes what the twin promises.

6. Cross-cutting concern II: retention, replay and provenance

6.1 Define a tier by the question it can answer

You know how to tier storage by latency and cost. The twin-specific move is to define each tier by **the question it can still answer**, treating the dropping of a field as a decision to stop being able to answer something. Figure 3 labels each tier that way, and the annotations between tiers are the losses you are choosing; read them as the design, not as the overflow.

Work each boundary from the questions rather than from the budget.

Live to replay. The surviving question is "why did the twin say that, then?". Answering it needs the inputs the twin actually consumed, at the resolution it consumed them, plus the release and schema identity. Full-rate retention of everything for eighteen months is

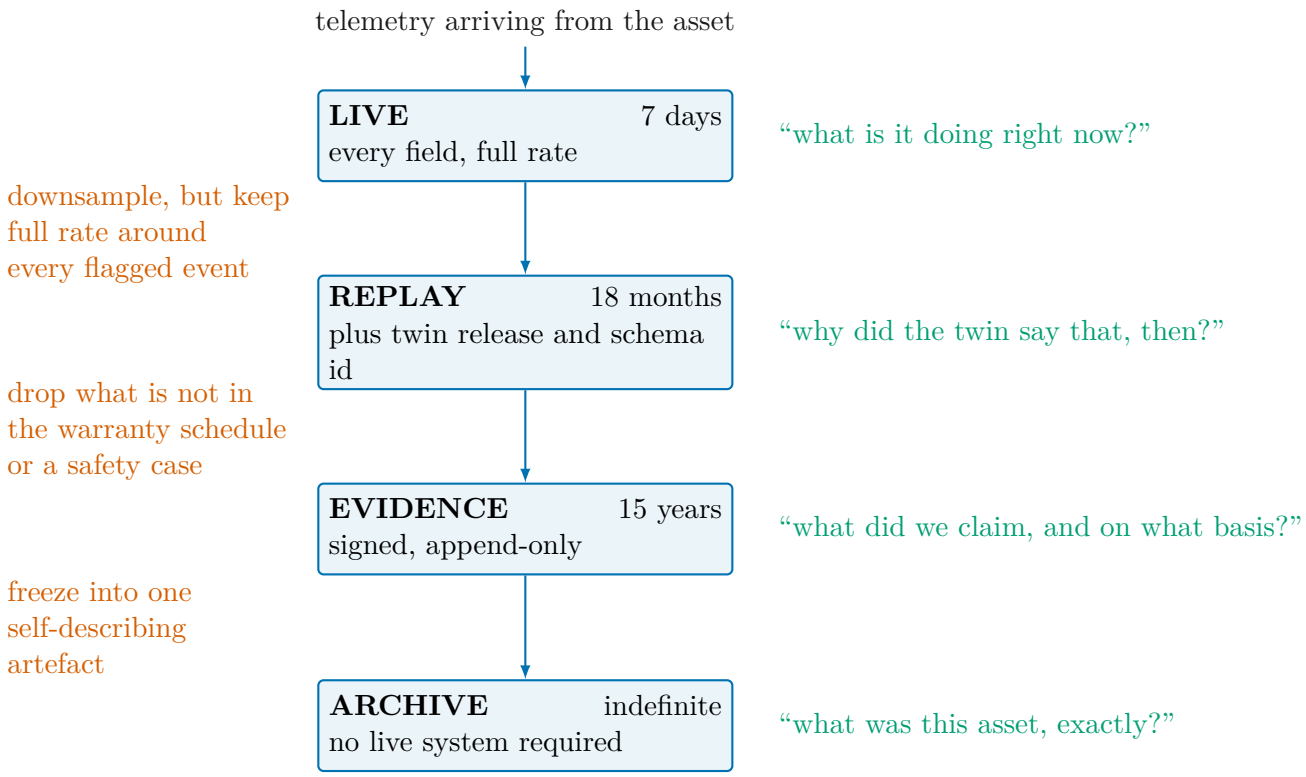


Figure 3: Four retention tiers for the storage fleet, each labelled with the question it preserves the ability to answer. The annotations between tiers are the deliberate losses.

unaffordable for 1,920 modules per site, so the fleet downsamples – keeping the full-rate window around every flagged event, because flagged events are what somebody asks about later.

Replay to evidence. The surviving question is "what did we claim, and on what basis?" – a far narrower set: the channels named in the warranty schedule and the safety case, plus the signed fidelity claims and revalidation results. Fifteen years of a narrow set is cheaper than eighteen months of everything.

Evidence to archive. The surviving question is "what was this asset, exactly?", asked by somebody with no access to any running system – a *format* problem rather than a storage class, treated in Section 6.4.

6.2 Replay decays silently

Replay – re-running a twin release over historical telemetry to reproduce a past result – is what makes most of the above worth paying for, and the capability most likely to be believed in and not to exist. It works only if four things hold at once: the telemetry is retained at sufficient resolution; the release that produced the original answer can still be built and run; the historical schema is known and the code that reads it still exists; and the calibration then in force is recoverable. Any one of the four rotting takes replay with it, and nothing reports it.

The mitigation is the one you would apply to any silently-degrading capability: **exercise it on a schedule.** Once a quarter, pick a stored answer at random from twelve months

ago, rebuild its release, replay its inputs, and check that the number matches. Treat a failure as an incident. This is cheap, and it is the only thing that turns "we can replay" from a belief into a fact.

6.3 Provenance makes an answer defensible

Provenance is the record of where a value came from: which sensor, which schema version, which twin release, which calibration. Work on representing twins as linked data argues for describing data sources explicitly, on the grounds that it is what keeps track of provenance and eases management of the supporting systems [31]. The engineering view is blunter: an answer without provenance cannot be defended, and a twin's most valuable answers are the ones that get challenged.

The honest counterweight is cost. Security surveys record that traceability including data provenance, together with auditing and accountability, presents serious storage problems given the volumes involved, alongside computational and communication overheads [25]. Full-fidelity provenance on every sample is unaffordable. The workable compromise records provenance at the granularity of *answers* rather than samples: every persisted output carries the triple and the calibration identity, and the inputs are recoverable through the retention tiers rather than duplicated per answer.

6.4 The archival record is a format problem

The archive tier is most often got wrong because it is treated as the cheapest storage class rather than a distinct artefact.

An archival record must be readable by someone with no access to your running systems, your database, your cloud account, or you – which rules out a database snapshot and a container image. It requires a self-describing bundle: data in an open, documented format; the schema, in the bundle; the model, exported through a standard interchange format where one exists; the calibration parameters and the data they were fitted from; the as-maintained configuration record; every signed fidelity claim and revalidation result; and a plain-language document saying what the twin was for and what its tolerances were.

Repository designs for twin data in construction engineering argue from the modelling side that models and the data they consume must be curated together, because either alone loses meaning [32]. The archival case takes that to its limit, where "together" means "in one bundle that outlives every system involved".

7. Cross-cutting concern III: interfaces that outlive their designers

7.1 A duration problem

You know how to evolve an interface. What differs is duration and counterparty: the twin's interface to the asset is negotiated with a control-system vendor whose product has a ten-year support life, on a site where an outage costs money, and it must survive until 2040.

Three interfaces age differently. **The telemetry interface** – asset to twin – is the one that breaks, changing when instrumentation is replaced or firmware upgraded, without

consulting you. **The command interface** – twin to asset, where one exists – is safety-relevant, and so the most tightly controlled and least likely to change silently. **The consumption interface** – twin to its consumers – is the one you own, and therefore the one you can version properly. Version it properly.

7.2 Use a standard as a fixed point, with clear eyes

Anchoring on a standard is good advice with a caveat. The manufacturing twin framework standard, ISO 23247, supplies a reference architecture and an information-exchange model, analysed in terms of the use cases it supports [33], [34]. In industrial automation, the Asset Administration Shell supplies a standardised electronic description of an asset for exchanging asset-related data between assets, production systems and engineering tools, and its relationship to twin concepts has been examined directly [35]. Interoperability frameworks argue that aligning data models, ontologies and terminology across consortia is what would reduce the integration cost keeping many use cases uneconomic [36].

The caveat is that the field's own reviews still find no agreed way of expressing a model, a data format, a connection, an interface's syntax or a domain ontology, and say so plainly [37]. Practitioner surveys find the same unevenness, with interfaces between twins expected to come later rather than being present now [12].

So: neither adoption nor rejection. **Adopt a standard as the vocabulary of your interface, and version your interface anyway.** The standard buys a shared meaning for a field; it does not guarantee the field will still mean that in 2035, and it does not relieve you of a compatibility policy.

7.3 A compatibility policy for a fifteen-year interface

Four rules, justified by duration rather than by general practice.

Schema versions are explicit in the payload, and the twin rejects unknown ones. Inferring the schema from the shape of the data is what lets a units change be absorbed silently. The refusal must be loud: a twin that stops answering is visibly broken, and a twin that answers wrongly is not.

Semantic changes are breaking changes, even when the syntax is unchanged. Reporting current in centi-amperes rather than amperes does not change the type; rebasing a temperature sensor's reference point does not change the type. Both invalidate every model fitted before them. Version on meaning, and keep a units declaration in the schema so the change is expressible at all.

Every interface has a stated support horizon and a named owner on both sides. In year seven a vendor's field engineer will ask whether anyone still consumes a particular tag, and somebody must be able to answer. An interface with no owner is discontinued by default.

Model interchange gets the same treatment as data interchange. A model locked inside a tool is an interface you cannot version. Export through a standardised exchange format where tooling supports it [19]; where it does not, record the obsolescence risk as a dependency with a date.

8. Cross-cutting concern IV: deployment topology and who is on call

8.1 Topology is a life-cycle decision

Where the twin runs is usually decided on latency and cost. Over fifteen years two other factors dominate.

Coupling to the asset's change control. Twin components deployed *on* the asset inherit the asset's change-control regime: a model inside the site controller is updated during an outage window with a permit, not on a Tuesday. Every component placed on the asset side moves from the fast clock to the slow one. The executable-twin idea – extracting a self-contained, deployable model that remains linked to the twin it came from – makes the trade explicit, and notes that the extracted artefact goes out of date unless somebody cuts a new one or builds it a traceable update path [38]. That "unless" is the whole life-cycle question.

Survivability of the platform. Fifteen years outlasts many managed services. Platform work aiming at portability – twins on general-purpose container orchestration with serverless scaling [39] – is attractive precisely because a portable target hedges against outliving your vendor, and open-source frameworks are worth weighing on the same axis, their failure mode being abandonment rather than withdrawal [40].

8.2 The fleet changes the arithmetic

Twelve sites make some things cheaper and one thing harder.

Cheaper: a fleet-level twin can detect one site diverging from its siblings, a far more sensitive drift detector than any single site's residuals; and calibration learned across a fleet transfers, which is a recognised motivation for fleet-level twins [41]. Probabilistic formulations of predictive twins are built so calibration and updating scale to a whole fleet rather than being redone per asset [42].

Harder: **twelve sites are twelve different asset configurations**, and they diverge as each is maintained separately. The compatibility triple is per site, so a release correct for eleven sites and wrong for the twelfth is the normal case, not the exception.

8.3 Operational ownership

The last concern is not technical. Who is paged when the twin is *wrong* – still running, still returning numbers, no longer fit for P1? Not the platform team, because it is not an outage. Not the data science team, because the model is doing what it was fitted to do. It is a life-cycle failure, and life-cycle failures belong to whoever owns the purpose table.

Make that ownership explicit, with three duties: hold the purpose table and approve changes to it; own the drift thresholds and the action ladder; sign revalidation results. Governance treatments argue that accountability and responsibility need clear lines established rather than assumed [11] – this is that case, reduced to three duties that fit in a role description.

8.4 Identity and the twin registry

You know service discovery. The twist is that a twin's identity has a physical referent, and physical referents get swapped, refurbished, relocated and renumbered by people who have never heard of your registry.

Three identities have to be modelled separately, and conflating any two causes a predictable failure:

- **Asset identity.** Which physical object this is. Survives relocation; does not survive replacement — the container installed in 2031 is a *different asset* from the one it replaced, and a registry that reuses the identifier makes fifteen years of history describe two containers as one.
- **Position identity.** Which slot in the installation this fills. "Container 7 at site 3" survives replacement and not re-arrangement. Usually what operators mean when they name a container, and usually not what the twin should key history on.
- **Twin instance identity.** Which running process. Changes on every deployment and should never be visible to a consumer.

The rule: **history keys on asset identity; operator interfaces key on position identity; nothing outside the twin sees instance identity.** The registry holds the time-indexed mapping between the first two — position P2 was asset 4471 from 2019-06 to 2024-03 and asset 8802 thereafter — which is what makes §6.7's six-week window representable at all, and what binding verifies against (§5.2). Tekinerdogan and Verdouw's pattern catalogue is a reasonable place to look for structures that separate these concerns, being organised around the systems engineering life cycle rather than a single deployment [43].

8.5 The human across the stages

The last cross-cutting concern is the one most often left out of an architecture, and the one that changes most across the life cycle.

Karnoub et al. argue that the human's role deserves explicit modelling rather than being left implicit, developing a digital triplet concept in which the human is a modelled participant alongside the physical and digital sides [44]. Shangguan et al. reach a related position from the architecture direction, proposing a triple human-digital-twin architecture on the grounds that a twin covering only some life-cycle phases cannot support collaboration across the whole of it [45].

You need adopt neither architecture to take the practical point: **the human's relationship to the twin differs at every stage, and a twin designed for one stage's human is wrong for the others**:

- At **conception**, the human is a domain expert whose knowledge the twin is trying to capture. The interface that matters is the one for stating and disputing assumptions.
- At **commissioning**, the human is a verifier. What they need is the ability to see the twin's raw disagreement with their instruments, unsmoothed.
- During **operation**, the human is a decision-maker under time pressure. What they need is the twin's confidence and its runtime phase, not its internals.
- During **evolution**, the human is a reviewer of changes they did not request and often cannot evaluate. What they need is a plain statement of what changed and what

evidence supports it — which is what the promotion record and the revalidation debt table exist to be.
- At **retirement**, the human is an archivist and, later, a litigant or an investigator. What they need is provenance.

A twin whose only human interface is the operating dashboard has designed for the middle of that list and abandoned both ends. The commissioning engineer of §6.6 who spotted the sensor-tapping difference was working at the second bullet; nothing in the twin's operating interface would have surfaced it.

9. Worked examples

Three examples, with the support deliberately fading. The first is worked completely, with the reasoning behind each choice visible. The second is worked up to a point and then hands you one step. The third is posed as a problem, with structure and grading criteria but no solution.

9.1 Worked Example 1 (fully worked): the revalidation trigger for P1

The problem. P1 is the annual state-of-health declaration, tolerance 1.5 percentage points absolute error on rated capacity. Design the mechanism that decides when the twin has drifted far enough that something must be done, and what that something is.

Step 1 – choose what to observe, and distrust the cheap signal.

The tempting observable is the twin's continuously estimated state of health, watched for a step or a trend. Reject it: that estimate is the twin's *output*, and comparing an output against itself measures nothing. Drift is a relationship between the twin and the asset, so the observable must involve a measurement of the asset the twin did not produce.

The fleet performs a **reference capacity test** on a rotating subset of containers each month: a controlled full discharge under specified conditions, giving a directly measured capacity. Use the **signed residual**:

```
r = SoH_predicted_by_twin - SoH_measured_by_reference_test
```

Why signed? A twin consistently over-predicting state of health produces an optimistic warranty declaration – a commercial and legal exposure. A twin under-predicting is conservative and costs money in unnecessary derating. Different problems, different owners; an absolute residual erases the distinction.

Step 2 – choose the statistic, and think about what corrupts it.

A single residual is noise. Use a **rolling median of the last six reference tests**.

Why median? A reference test occasionally fails in a way that produces a wild value – an aborted discharge, a thermal interlock, an operator error. A mean over six points moves substantially on one such value; a median does not. The trigger should fire on drift, not on a failed test.

Why six? Six monthly tests on a rotating subset span roughly half a year per container: long enough to average test-to-test noise, short enough that genuine drift is not diluted. It is a judgment, and the parameter most worth revisiting after two years of data.

Step 3 – set the threshold from the purpose, not from the data.

This step is most often done backwards. The common approach inspects the residual distribution, takes three standard deviations, and calls that a threshold – which describes how the twin *has* behaved, not how much error the *decision* can absorb. Work down from the tolerance instead:

Table 2: Decomposing the P1 tolerance into a drift budget. Every line above the last is spent before the twin has drifted at all.

Budget line	Allocation
P1 tolerance, total	1.5 pp
Reserved for reference-test measurement error	0.5 pp
Reserved for estimator uncertainty at declaration time	0.5 pp
Remaining headroom for observed drift	**0.5 pp**

Read Table 2 downwards: two thirds of the tolerance is consumed by measurement and estimator error, neither of which drift causes and neither of which drift can be allowed to reuse. The drift budget is 0.5 pp, so the trigger must fire below it with enough margin to act before the budget is spent. Set the primary trigger at **0.3 pp** and treat 0.5 pp as the point at which the P1 claim is no longer supportable.

Note what happened: the threshold came from a contract and two error budgets, with no residual data consulted. The data tells you whether the threshold is *achievable*, a separate and later question. If observed residuals sit at 0.4 pp, the answer is not to relax the threshold; it is to improve the reference test, improve the estimator, or renegotiate P1.

Step 4 – remove the confound before it embarrasses you.

Capacity measurement is temperature-dependent and the sites are outdoors. Twelve months of median signed residuals from one container, in percentage points:

```
Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov  Dec
0.31 0.29 0.18 0.09 0.04 0.02 0.03 0.06 0.11 0.21 0.28 0.33
```

A flat 0.3 pp threshold fires in January, November and December, every year, on every container, for as long as the site exists. Within two winters the alert is muted and the mechanism is dead.

Two responses are defensible and not equally good. *Raising the threshold to 0.35 pp* spends the drift budget accommodating a known, modelled effect, and hides genuine drift of up to 0.35 pp in summer. Reject it. *Adjusting for the confound* – including measured cell temperature at test time as a covariate and thresholding the adjusted residual – is better: the seasonal component is physical, understood and measured, so leaving it in the signal is a choice to detect it repeatedly. After adjustment the series flattens to roughly 0.02 to 0.06 pp and the 0.3 pp threshold becomes meaningful.

The lesson generalises beyond batteries: **before setting a drift threshold, enumerate the periodic and operating-regime effects that will move the residual for reasons that are not drift, and remove them explicitly.** A threshold that fires predictably will be ignored.

Step 5 – build the action ladder.

One threshold with one action is not enough, because the available actions differ in cost by two orders of magnitude. Build a ladder, with a named owner and a deadline on every rung.

Table 3: The action ladder for P1: each band of adjusted residual, the response it obliges, who owns it and what it costs.

Adjusted residual	Action	Owner	Cost	Deadline
below 0.15 pp	Record. No action.	automatic	none	–
0.15 to 0.30 pp	Raise a tracked observation; check sibling sites for the same movement.	twin team on call	hours	5 working days
0.30 to 0.50 pp	**Recalibrate**: re-fit parameters on the last 12 months, re-run the acceptance suite, release.	twin team	about 1 day	15 working days
above 0.50 pp	**Revalidate**: full re-examination of fitness for P1, including whether the model form is still right. May require asset testing.	purpose-table owner, with engineering sign-off	2 to 4 weeks	30 working days

above 0.50 pp, revalidation past deadline	**Suspend P1**: the twin stops answering the state-of-health question and returns a refusal; declaration falls back to direct measurement on every container.	purpose-table owner	very high	immediate

Read the cost column of Table 3 against the residual column: cost rises by roughly an order of magnitude per rung, which is why one threshold with one action is the wrong shape. Three features are worth arguing for explicitly.

The bottom rung is "record", not "nothing". You want the history of sub-threshold movement, because the interesting signal is often a slow climb through the quiet band rather than a jump into the loud one.

Sibling comparison sits on the second rung, not the fourth. Drift on one container is a container problem; identical drift across all twelve sites is a twin problem, most likely a schema or code change. Asking early costs minutes and routes the incident correctly.

The top rung exists and is exercised. A ladder whose top rung has never been used is a ladder whose top rung does not work. Rehearse the fall-back annually on one site.

Step 6 – state what this does not catch.

Every drift detector has blind spots, and they belong written down beside the detector. P3, which has no routine reference test, is not covered by this mechanism at all – record that as a known limitation rather than leaving it to be discovered.

Common-mode error. If a change corrupts the twin's estimate and the reference test in the same direction – firmware altering both the current the twin uses and the current used to integrate the reference discharge – the residual stays near zero while both are wrong. The defence is outside the mechanism: independent instrumentation on the reference test, and schema-change detection from Section 5.

Slow model-form error. Recalibration re-fits parameters within a fixed model form. If the form itself is becoming wrong – a degradation mechanism appears that the model has no term for – recalibration keeps the residual small for a while by distorting parameters, and the failure surfaces later as a sudden loss of predictive power outside the recent operating range. That is why the top rung asks whether the model form is still right, and why recalibration must not run unboundedly without periodic revalidation. Reviews of twin methodology treat identifying and quantifying both parameter and model-discrepancy uncertainty as an activity distinct from fitting, for exactly this reason [6].

9.2 Worked Example 2 (support faded): a firmware upgrade changes a unit

The situation. The battery management system vendor releases firmware 4.2. Among many changes, it reports pack current in centi-amperes rather than amperes and adds a field for cell-balancing state. The change is documented in a release note nobody on the twin team receives. Site 7 is upgraded during a maintenance window on 12 March; sites 1 to 6 and 8 to 12 are scheduled for the following quarter.

Steps 1 to 4 are worked below. Step 5 is yours.

Step 1 – what breaks, and how loudly?

Pack current is off by a factor of 100. Every derived quantity is wrong: throughput, coulomb-counted state of charge, and the P2 day-ahead energy prediction. State of health is affected through its dependence on integrated charge. Does anything catch it?

- *Type checking:* no. The field is still numeric and still positive.
- *Range checking:* possibly. A hundredfold increase exceeds a tight physical bound and passes a loose one – an argument for bounds taken from nameplate ratings rather than observed data.
- *The P1 drift trigger:* eventually, at site 7's next reference test. That means up to a month, and it will point at the model rather than the interface.
- *Sibling comparison:* immediately. Site 7 diverging from eleven siblings the day after a maintenance window is a near-unambiguous signature, which is the point of the ladder's second rung.

Step 2 – what should have caught it?

The schema contract from Section 7.3. With an explicit schema version in the payload and rejection of unknown versions, the twin refuses at 09:00 on 12 March, and the incident is a visible outage of one site's twin rather than a silent corruption of one site's history. That is a *worse-looking* outcome and a much better one. The trade will be argued with you: refusing costs a day of one site's twin availability; answering wrongly costs an unknown number of contaminated records discovered at an unknown later date, and that cost is paid by someone not in the room when the decision is made.

Step 3 – what must the twin release do?

The twin must read two telemetry formats, because most sites will not be upgraded for a quarter. So the release: declares support for both schema versions; normalises both to a single internal representation with declared units, at the ingestion boundary and nowhere else; records the schema version alongside every stored answer; and retains the new cell-balancing field in raw storage even though no model consumes it yet, because retention decisions are hard to reverse and this one is cheap.

Step 4 – what does the triple look like now?

For four months the fleet occupies two triples simultaneously:

```
site 7:            (config A2, release R5, schema S2)
sites 1-6, 8-12:   (config A1, release R5, schema S1)
```

Release R5 is compatible with both. This is the normal steady state of a fleet mid-rollout, and it is why the triple is recorded per site and per answer rather than per fleet.

Step 5 – your step. Between 12 March and discovery on 2 April, site 7 wrote 21 days of answers into the replay tier and the evidence tier. Decide what happens to them. Address four questions:

- Which tier requires action, and is it the same action in each?
- The raw telemetry for those 21 days is retained and is not itself corrupted – correct centi-amperes, wrongly interpreted. What follows?
- Should affected answers be corrected in place, superseded, or deleted? Argue from the properties the evidence tier must have, not from convenience.
- What has to be true about how those answers were stored for your remedy to be possible at all?

Hint for the third question: consider what the evidence tier is for – supporting a claim that may be challenged years later – and what a challenger would conclude on discovering that records in it can be edited.

The shape of a good answer, once you have tried it: the two tiers need different treatments because they have different properties; the retained raw telemetry is what makes any remedy possible, and retaining it is the decision that saved you; and the choice between correction and supersession is settled by the fact that an append-only evidence tier is far more defensible than an editable one, which pushes you towards superseding records that name what they supersede and why. All of it depends on the answers having been stored with their schema version attached – which is how you knew which 21 days were affected rather than guessing from dates.

9.3 Worked Example 3 (posed as a problem): retiring a twin at repowering

The situation. Site 3, commissioned in 2023, is repowered in 2031: cells replaced with a different chemistry from a different supplier; containers and thermal management replaced; inverters retained; site control system upgraded by two major releases. The grid connection, site identity and grid-services contract are unchanged. The original capacity warranty ends with the old cells; a new warranty begins with the new ones.

Your task. Produce two documents.

1. **A retirement plan for the existing site-3 twin**: what is decommissioned and when; what is archived and in what form; what is destroyed and on whose authority; which obligations survive.
2. **An inheritance specification for the successor twin**: what it takes from the old one, what it must not take, and what it must establish afresh.

Grading criteria, each mapping to a section of this chapter.

- *It separates the three endings.* This is a repowering, not an asset retirement and not a sale. Say which of Section 4.5's cases applies and what follows.
- *It identifies which obligations outlive the twin, and for how long.* The old warranty has ended, but claims under it may not have; the fifteen-year evidence horizon started in 2023 and does not restart.
- *It is explicit about the calibration history*, which has no predictive value for the new chemistry and considerable evidential value for the old warranty. A plan treating it

as one thing is wrong.

- *It settles what the successor may inherit.* Some things transfer cleanly: electrical topology, the grid connection model, the operating-regime history, the purpose table's P2 and P3 rows. Some must not: any calibration fitted to the old chemistry, any drift threshold derived from it, any fidelity claim. Say which, and why.
- *It names a new C3*, with a date, an author, the measurements it rests on, and a statement of what the twin may answer in the interim.
- *It treats the control-system upgrade separately from the cell replacement.* One invalidates models, the other interfaces. A plan treating the repowering as a single event misses the interface work.
- *It says what is destroyed.* An archived twin of a live site describes a facility that still exists. Which parts become a liability once they are no longer an asset?

Where to start: draw the two clocks for site 3 across 2031 and mark the coupling points. The repowering fires C4 and C6 for the old twin and C1 through C3 for the new one, within a few months. The retirement plan is the C6 deliverable, the inheritance specification the C1 deliverable, and separating them makes most remaining decisions fall out.

10. Exercises

Seven exercises, each naming the objective it exercises. **Solutions in outline are given for Exercises 1, 3 and 5; the rest have hints only.** Difficulty: (E) straightforward application, (M) requires a judgment, (H) open-ended.

Exercise 1 (E) [Objective 1]. A sensor on one container fails and is replaced with a functionally equivalent unit from a different supplier, mounted 40 cm from the original position. Classify this as parametric or structural, identify which coupling point it fires, and state what the twin team owes as a result.

> *Solution in outline.* Parametric with respect to the twin's architecture – no component added or removed, topology unchanged – but not a no-op, because the mounting position changes what the sensor measures and the new supplier's transfer characteristics may differ. It fires C4. The twin team owes an update to the asset configuration record, incrementing the site's configuration identifier; a check of whether any calibration used that sensor's history, since a discontinuity has entered a series the model may treat as continuous; and a decision on whether that discontinuity requires recalibration. The subtle part is the third: a change that is structurally parametric can still be statistically significant, and the two classifications answer different questions.

Exercise 2 (M) [Objective 2]. Design the drift trigger for P3, the thermal envelope check. P3 has no monthly reference test. State what you would observe instead, what your threshold would be, and what your design cannot detect.

> *Hint.* Worked Example 1 relied on an independent measurement of the predicted quantity. Ask what one exists for peak module temperature during normal operation, and note that P3's required *conservative* bias changes what a good residual distribution looks like: a one-sided test is not a mistake here.

Exercise 3 (E) [Objective 3]. A release is deployed to all twelve sites. Two weeks

later it is found correct for eleven and wrong for site 9, which had a container removed for repair in a way the release did not account for. Using the compatibility triple, state exactly what went wrong and what would have prevented it.

> *Solution in outline.* Site 9's asset configuration had moved and the release did not declare support for the new configuration – but nothing checked, because the release declared support for a *fleet* rather than an enumerated set of configurations. The triple was being tracked at the wrong granularity. The preventive mechanism is Section 5.4's second rule: a release declares the configurations it supports and refuses anything else, so site 9's twin would have refused on deployment – an obvious, immediately-noticed failure. Note the secondary failure: the container removal should have fired C4 and produced a twin-team review. Two independent defences both absent is the usual shape of such an incident.

Exercise 4 (H) [Objective 4]. Finance proposes reducing the replay tier from 18 months to 6, saving a stated sum per year. Write the response. It should not be a refusal; it should state precisely which questions become unanswerable and what the fall-back is for each.

> *Hint.* Figure 3 labels each tier with the question it preserves. Enumerate the questions actually asked of the replay tier in the last two years and separate those the evidence tier can serve. A good response probably proposes a different cut: narrower channels for longer, rather than everything for less time.

Exercise 5 (M) [Objective 4]. Design the quarterly replay exercise from Section 6.2. Specify what is selected, how, what constitutes a pass, and what happens on failure.

> *Solution in outline.* Select a stored answer at random from the window 12 to 15 months ago, weighted towards answers consumed by a decision rather than merely computed. Rebuild the release named in the answer's triple from source; fetch inputs from the replay tier using the schema version named in the triple; re-run. A pass is bit-identical output, or output within a documented numerical tolerance if the computation is not deterministic – and being unable to say which applies is itself the finding. On failure, raise an incident and classify the cause into four buckets: data no longer retained, release no longer buildable, schema no longer readable, calibration no longer recoverable. The classification matters more than the individual failure, because each bucket has a different systemic fix and each will recur. Record results where the trend is visible: the useful signal is a rising failure rate over years, which is what silent decay looks like.

Exercise 6 (M) [Objective 5]. Site 11 is sold to another operator who wants the twin. List what you would transfer, withhold, and renegotiate before transferring anything. For each withheld item, say what makes it withholdable.

> *Hint.* Three categories appear: things that are the asset's and go with it; things that are yours and encode capability developed across the fleet; and things that are a third party's, licensed to you rather than owned. The middle is where the argument happens, and fleet-derived calibration sits squarely in

it.

Exercise 7 (H) [Objectives 1 and 5]. A twin is commissioned for an asset that does not yet exist: it is used during design and virtual commissioning, and the physical asset is built two years later. Redraw Figure 1 for this case. Which coupling points move, which change meaning, and what new risk appears that the original does not show?

> *Hint.* Section 3.3 noted that a twin validated against a design has made a different claim from one validated against a built asset. Follow that through: the risk is a claim made at one coupling point being read as though made at a different one, two years apart, by people present for neither.

Five exercises were drafted and cut, and are recorded in the dossier's `rejected.md` with reasons – among them one asking the reader to implement a schema-registry integration, which is a keyboard task rather than a chapter exercise, and one asking for a comparison of commercial twin platforms, which Section 1.3 excludes. Every objective still carries at least one exercise.

11. Summary, and where to go next

11.1 What this chapter established

Two clocks, six coupling points (Objective 1). The asset and the twin have separate life cycles, owners and cadences. Modelling them as one process throttles the fast clock to the slow one. Modelling them as two, joined at explicit coupling points, tells you whose work is owed when, when you may move fast, and which artefacts constitute the twin's audit trail.

Drift is a maintenance obligation with a period (Objective 2). A twin becomes unfit slowly while continuing to look healthy. Catching it needs an independent observable, a statistic robust to bad tests, a threshold derived from the purpose rather than the data, explicit removal of confounds, and an action ladder with an owner on every rung – including a rehearsed top rung. Every detector has a blind spot, and it belongs written down beside the detector.

The compatibility triple (Objective 3). Asset configuration, twin release and telemetry schema move independently. The goal is not to synchronise them, which is unachievable, but to make disagreement detectable and the twin refuse rather than extrapolate. Every stored answer carries all three.

Retention tiers are defined by questions (Objective 4). Not by latency and not by price. Dropping a field is a decision to stop being able to answer something. Replay justifies most of the cost, decays without signalling, and is defended only by exercising it on a schedule.

Retirement is a phase with deliverables (Objective 5). Three endings – asset retired, twin replaced, asset sold – with different deliverables. Retention and security obligations both outlive the twin, and the archival record is a format problem rather than a storage-class one: it must be readable by someone with access to none of your systems.

Underneath all five is one claim worth carrying away. **The expensive questions a twin answers are asked late in its life, and a twin can only answer them if it was built early to still be able to.**

11.2 Where to go next

The discipline as a whole. Book-length treatments of engineering twins for cyber-physical systems [46], [47], with case studies collected separately [48], a hands-on tutorial case [49], and foundational concepts including the fleet-level framing used here [41].

The process framing this chapter compressed. The mapping of systems-engineering technical processes onto twin-enabled systems [8], with the V-model spanning conception to decommissioning as its complement [7]. Broader life-cycle surveys situate both [50], [51].

Automating more of the release path. The declarative, version-controlled approach [26] and its precursor experiments [52]; the model-based operations literature [28]–[30]; and the interview evidence on where such automation stops [3].

Structural change. The separation of behavioural from architectural self-adaptation [15], with a worked exemplar [53] and a reconfiguration-management framework [18].

Drift and uncertainty. The two-part comprehensive review is the entry point – Part 1 on modelling and twinning technologies, Part 2 on uncertainty quantification and the battery case [6], [54]. The research-gaps assessment is the best statement of fitness for purpose as the organising question [9], and the scaling argument sets the ambition [2]. For the case study's domain, probabilistic battery health prognostics [55] and model-based remaining-useful-life work [56].

Consulted but not cited above: predictive maintenance using twins, the service most twins are eventually asked for [57]; a twin evolving in parallel with its physical counterpart over a real deployment [58]; a full-scale validation against a physical prototype, a good model of a signed fidelity claim in practice [59]; the security and privacy issues a twin raises [60]; and practitioners describing what they actually got from twins across a building's life cycle [61].

12. References

[1] N. Zhang, R. Bahsoon, N. Tziritas, and G. Theodoropoulos, "Knowledge Equivalence in Digital Twins of Intelligent Systems," *ACM Transactions on Modeling and Computer Simulation*, vol. 34, no. 1, pp. 1–37, Jan. 2024, doi: 10.1145/3635306.

[2] S. A. Niederer, M. S. Sacks, M. Girolami, and K. Willcox, "Scaling digital twins from the artisanal to the industrial," *Nature Computational Science*, vol. 1, no. 5, pp. 313–320, May 2021, doi: 10.1038/s43588-021-00072-5.

[3] F. Zampetti, D. Tamburri, S. Panichella, A. Panichella, G. Canfora, and M. Di Penta, "Continuous Integration and Delivery Practices for Cyber-Physical Systems: An Interview-Based Study," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 3, pp. 73:1–73:44, Apr. 2023, doi: 10.1145/3571854.

[4] J. Michael *et al.*, "Model-Driven Engineering for Digital Twins: Opportunities and Challenges," *Systems Engineering*, vol. 28, no. 5, pp. 659–670, 2025, doi: 10.1002/sys.21815.

[5] U. T. Tygesen, K. Worden, T. Rogers, G. Manson, and E. J. Cross, "State-of-the-Art and Future Directions for Predictive Modelling of Offshore Structure Dynamics Using Machine Learning," in *Dynamics of Civil Structures, Volume 2*, S. Pakzad, Ed., Cham: Springer International Publishing, 2019, pp. 223–233. doi: 10.1007/978-3-319-74421-6_30.

[6] A. Thelen *et al.*, "A comprehensive review of digital twin—part 2: Roles of uncertainty quantification and optimization, a battery digital twin, and perspectives," *Structural and Multidisciplinary Optimization*, vol. 66, no. 1, p. 1, Dec. 2022, doi: 10.1007/s00158-022-03410-x.

[7] R. Honcak and A. Wooley, "An MBSE approach for Virtual Verification & Validation of Systems with Digital Twins," in *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, Linz Austria: ACM, Sep. 2024, pp. 390–400. doi: 10.1145/3652620.3688252.

[8] J. Fitzgerald, K. Pierce, and K. Kristensen, "Digital Twin Engineering Processes," in *The Engineering of Digital Twins*, J. Fitzgerald, C. Gomes, and P. G. Larsen, Eds., Cham: Springer International Publishing, 2024, pp. 65–86. doi: 10.1007/978-3-031-66719-0_4.

[9] *Foundational Research Gaps and Future Directions for Digital Twins*. Washington, D.C.: National Academies Press, 2024. doi: 10.17226/26894.

[10] T. R. Wanasinghe *et al.*, "Digital Twin for the Oil and Gas Industry: Overview, Research Trends, Opportunities, and Challenges," *IEEE Access*, vol. 8, pp. 104175–104197, 2020, doi: 10.1109/ACCESS.2020.2998723.

[11] O. J. Pinon Fischer, S. Sabri, and Y. Chen, "Fundamentals of Digital Twins, Modeling Approaches, and Governance," in *Digital Twin: Fundamentals and Applications*, S. Sabri, K. Alexandridis, and N. Lee, Eds., Cham: Springer Nature Switzerland, 2024, pp. 13–29. doi: 10.1007/978-3-031-67778-6_2.

[12] E. Ferko, A. Bucaioni, P. Pelliccione, and M. Behnam, "Standardisation in Digital Twin Architectures in Manufacturing," in *2023 IEEE 20th International Conference on Software Architecture (ICSA)*, Mar. 2023, pp. 70–81. doi: 10.1109/ICSA56044.2023.00015.

[13] T. Li *et al.*, "Coupling a capacity fade model with machine learning for early prediction of the battery capacity trajectory," *Applied Energy*, vol. 389, p. 125703, Jul. 2025, doi: 10.1016/j.apenergy.2025.125703.

[14] M. Frasheri *et al.*, "Addressing time discrepancy between digital and physical twins," *Robotics and Autonomous Systems*, vol. 161, p. 104347, Mar. 2023, doi: 10.1016/j.robot.2022.104347.

[15] E. Kamburjan, N. Bencomo, S. L. Tapia Tarifa, and E. B. Johnsen, "Declarative Lifecycle Management in Digital Twins," in *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, Linz Austria: ACM, Sep. 2024, pp. 353–363. doi: 10.1145/3652620.3688248.

[16] M. Frasheri, T. Böttjer, P. G. Larsen, L. Esterle, and C. Gomes, "Advanced Digital Twin Services," in *The Engineering of Digital Twins*, J. Fitzgerald, C. Gomes, and P. G. Larsen, Eds., Cham: Springer International Publishing, 2024, pp. 209–222. doi: 10.1007/978-3-031-66719-0_10.

[17] L. Esterle, M. Frasheri, and P. G. Larsen, "Autonomous Reconfiguration Enabled by Digital Twins," in *The Engineering of Digital Twins*, J. Fitzgerald, C. Gomes, and P. G. Larsen, Eds., Cham: Springer International Publishing, 2024, pp. 345–362. doi: 10.1007/978-3-031-66719-0_14.

[18] B. Caesar, K. Barton, D. Tilbury, and A. Fay, "Digital Twin Framework for Reconfiguration Management: Concept & Evaluation," *IEEE Access*, vol. PP, pp. 1–1, Jan. 2023, doi: 10.1109/ACCESS.2023.3331221.

[19] G. Schweiger *et al.*, "An empirical survey on co-simulation: Promising standards, challenges and research needs," *Simulation Modelling Practice and Theory*, vol. 95, pp. 148–163, Sep. 2019, doi: 10.1016/j.simpat.2019.05.001.

[20] D. Dittler, P. Lierhammer, D. Braun, T. Müller, N. Jazdi, and M. Weyrich, "An Agent-based Realisation for a continuous Model Adaption Approach in intelligent Digital Twins." arXiv, Dec. 2022. doi: 10.48550/arXiv.2212.03681.

[21] C. Lu, Q. Xu, T. Yue, S. Ali, T. Schwitalla, and J. F. Nygård, "EvoCLINICAL: Evolving Cyber-Cyber Digital Twin with Active Transfer Learning for Automated Cancer Registry System," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, Nov. 2023, pp. 1973–1984. doi: 10.1145/3611643.3613897.

[22] Q. Xu, T. Yue, S. Ali, and M. Arratibel, "Pretrain, Prompt, and Transfer: Evolving Digital Twins for Time-to-Event Analysis in Cyber-physical Systems." arXiv, Apr. 2024. doi: 10.48550/arXiv.2310.00032.

[23] M. Grieves and J. Vickers, "Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems," in *Transdisciplinary Perspectives on Complex Systems: New Findings and Approaches*, F.-J. Kahlen, S. Flumerfelt, and A. Alves, Eds., Cham: Springer International Publishing, 2017, pp. 85–113. doi: 10.1007/978-3-319-38756-7_4.

[24] E. Altamiranda, "A System of Systems Foundation for Digital Asset Lifecycle Management," in *Digital Twin: Fundamentals and Applications*, S. Sabri, K. Alexandridis, and N. Lee, Eds., Cham: Springer Nature Switzerland, 2024, pp. 59–87. doi: 10.1007/978-3-031-67778-6_4.

[25] C. Alcaraz and J. Lopez, "Digital Twin: A Comprehensive Survey of Security Threats," *IEEE Communications Surveys & Tutorials*, vol. 24, no. 3, pp. 1475–1503, 2022, doi: 10.1109/COMST.2022.3171465.

[26] G. Beaumont, A. Beugnard, S. Martínez, C. Urtado, and S. Vauttier, "Towards Automating the Life Cycle Management of Digital Twins," in *ER2025-44th International Conference on Conceptual Modeling*, 2025. Accessed: Oct. 22, 2025. [Online]. Available: https://hal.science/hal-05225783/

[27] F. Beetz and S. Harrer, "GitOps: The Evolution of DevOps?" *IEEE Software*, vol. 39, no. 4, pp. 70–75, Jul. 2022, doi: 10.1109/MS.2021.3119106.

[28] B. Combemale *et al.*, "Model-Based DevOps: Foundations and Challenges," in *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, Oct. 2023, pp. 429–433. doi: 10.1109/MODELS-C59198.2023.00076.

[29] J. Hugues, J. Yankel, J. Hudak, and A. Hristozov, "Twinops: Digital twins meets devops," *CARNEGIE-MELLON UNIV PITTSBURGH PA, Tech. Rep.*, 2022, Accessed: Jan. 08, 2025. [Online]. Available: https://apps.dtic.mil/sti/trecms/pdf/AD1168471.pdf

[30] S. Aissat, J. Beaulieu, É. Poirier, A. Motamedi, J. Gascon-Samson, and F. Bordeleau, "A devops framework for the systematic engineering and evolution of digital twins for built assets," *Software and Systems Modeling*, Oct. 2025, doi: 10.1007/s10270-025-01330-0.

[31] S. Burattini, A. Zimmermann, M. Picone, and A. Ricci, "Towards Linked Data for Ecosystems of Digital Twins," in *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, in MODELS Companion '24. New York, NY, USA: Association for Computing Machinery, Oct. 2024, pp. 332–337. doi: 10.1145/3652620.3688245.

[32] P. Zech, P. Pobitzer, G. Fröch, and R. Breu, "A Proposal for a Models-Meet-Data Repository for Digital Twins in Construction Engineering," in *2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C)*, Jun. 2024, pp. 111–118. doi: 10.1109/ICSA-C63560.2024.00027.

[33] G. Shao, "Use Case Scenarios for Digital Twin Implementation Based on ISO 23247," *NIST*, May 2021, Accessed: Mar. 06, 2024. [Online]. Available: https://www.nist.gov/publications/use-case-scenarios-digital-twin-implementation-based-iso-23247

[34] G. Shao, S. Frechette, and V. Srinivasan, "An Analysis of the New ISO 23247 Series of Standards on Digital Twin Framework for Manufacturing," American Society of Mechanical Engineers Digital Collection, Sep. 2023. doi: 10.1115/MSEC2023-101127.

[35] J. Zhang, C. Ellwein, M. Heithoff, J. Michael, and A. Wortmann, "Digital twin and the asset administration shell," 2024, Accessed: Apr. 09, 2025. [Online]. Available: https://awortmann.github.io/downloads/paper/Digital_twin_and_the_asset_administration_shell.pdf

[36] A. Budiardjo and D. Migliori, "Digital twin system interoperability framework," Tech. rep. Digital Twin Consortium, East Lansing, Michigan, 2021. Accessed: Feb. 25, 2025. [Online]. Available: https://www.digitaltwinconsortium.org/pdf/Digital-Twin-System-Interoperability-Framework-12072021.pdf

[37] N. Anwer, R. Stark, F. Tao, and J. Erkoyuncu, "Developing and leveraging digital twins in engineering design," *CIRP Annals*, vol. 2025, Jun. 2025, doi: 10.1016/j.cirp.2025.05.002.

[38] D. Hartmann and H. Van der Auweraer, *The Executable Digital Twin: Merging the digital and the physics worlds*. 2022.

[39] A. G. Wermann and J. A. Wickboldt, "KTWIN: A Serverless Kubernetes-based Digital Twin Platform." arXiv, Aug. 2024. doi: 10.48550/arXiv.2408.01635.

[40] S. Gil, P. H. Mikkelsen, C. Gomes, and P. G. Larsen, "Survey on open-source digital twin frameworks–A case study approach," *Software: Practice and Experience*, vol. 54, no. 6, pp. 929–960, Jun. 2024, doi: 10.1002/spe.3305.

[41] C. Gomes, B. J. Oakes, J. Fitzgerald, and P. G. Larsen, "Foundational Concepts for Digital Twins of Cyber-Physical Systems," in *The Engineering of Digital Twins*, J. Fitzgerald, C. Gomes, and P. G. Larsen, Eds., Cham: Springer International Publishing, 2024, pp. 45–63. doi: 10.1007/978-3-031-66719-0_3.

[42] M. G. Kapteyn, J. V. R. Pretorius, and K. E. Willcox, "A probabilistic graphical model foundation for enabling predictive digital twins at scale," *Nature Computational Science*, vol. 1, no. 5, pp. 337–347, May 2021, doi: 10.1038/s43588-021-00069-0.

[43] B. Tekinerdogan and C. Verdouw, "Systems Architecture Design Pattern Catalog for Developing Digital Twins," *Sensors*, vol. 20, no. 18, p. 5103, Jan. 2020, doi: 10.3390/s20185103.

[44] A. Karnoub, D. Stuber, J. Mertes, M. Klar, and L. Yi, "How to understand the role of human in digital twins: A digital triplet concept," *Digital Engineering*, vol. 6, p. 100052, Sep. 2025, doi: 10.1016/j.dte.2025.100052.

[45] D. Shangguan, L. Chen, C. Su, J. Ding, and C. Liu, "A Triple Human-Digital Twin Architecture for Cyber-Physical Systems," *Computer Modeling in Engineering & Sciences*, vol. 131, no. 3, pp. 1557–1578, 2022, doi: 10.32604/cmes.2022.018979.

[46] J. Fitzgerald, C. Gomes, and P. G. Larsen, Eds., *The Engineering of Digital Twins*. Cham: Springer International Publishing, 2024. doi: 10.1007/978-3-031-66719-0.

[47] P. G. Larsen, J. Fitzgerald, and C. Gomes, "Engineering Digital Twins for Cyber-Physical Systems," in *The Engineering of Digital Twins*, J. Fitzgerald, C. Gomes, and P. G. Larsen, Eds., Cham: Springer International Publishing, 2024, pp. 3–17. doi: 10.1007/978-3-031-66719-0_1.

[48] B. J. Oakes *et al.*, "Case Studies in Digital Twins," in *The Engineering of Digital Twins*, J. Fitzgerald, C. Gomes, and P. G. Larsen, Eds., Cham: Springer International Publishing, 2024, pp. 257–310. doi: 10.1007/978-3-031-66719-0_12.

[49] C. Gomes *et al.*, "Digital Twin Tutorial: The Incubator Case Study," in *Engineering Trustworthy Software Systems: 6th International School, SETSS 2024, Chongqing, China, April 14–21, 2024, Tutorial Lectures*, J. P. Bowen, C. Gomes, and Z. Liu, Eds., Singapore: Springer Nature, 2025, pp. 68–101. doi: 10.1007/978-981-96-4656-2_3.

[50] C. Semeraro, M. Lezoche, H. Panetto, and M. Dassisti, "Digital twin paradigm: A systematic literature review," *Computers in Industry*, vol. 130, p. 103469, Sep. 2021, doi: 10.1016/j.compind.2021.103469.

[51] K. Y. H. Lim, P. Zheng, and C.-H. Chen, "A state-of-the-art survey of Digital Twin: Techniques, engineering product lifecycle management and business innovation perspectives," *Journal of Intelligent Manufacturing*, vol. 31, no. 6, pp. 1313–1337, Aug. 2020, doi: 10.1007/s10845-019-01512-w.

[52] G. Beaumont, A. Beugnard, S. Martínez, C. Urtado, and S. Vauttier, "Towards Re-Engineering Digital Twins: Preliminary Experiments on Three Use Cases," in *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, in MODELS Companion '24. New York, NY, USA: Association for Computing Machinery, Oct. 2024, pp. 453–458. doi: 10.1145/3652620.3688259.

[53] E. Kamburjan *et al.*, "GreenhouseDT: An Exemplar for Digital Twins," in *Proceedings of the 19th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, in SEAMS '24. New York, NY, USA: Association for Computing Machinery, Jun. 2024, pp. 175–181. doi: 10.1145/3643915.3644108.

[54] A. Thelen *et al.*, "A comprehensive review of digital twin — part 1: Modeling and twinning enabling technologies," *Structural and Multidisciplinary Optimization*, vol. 65, no. 12, p. 354, Nov. 2022, doi: 10.1007/s00158-022-03425-4.

[55] A. Thelen, X. Huan, N. Paulson, S. Onori, Z. Hu, and C. Hu, "Probabilistic machine learning for battery health diagnostics and prognostics—review and perspectives," *npj Materials Sustainability*, vol. 2, no. 1, pp. 1–33, Jun. 2024, doi: 10.1038/s44296-024-00011-1.

[56] A. Thelen, M. Li, C. Hu, E. Bekyarova, S. Kalinin, and M. Sanghadasa, "Augmented model-based framework for battery remaining useful life prediction," *Applied Energy*, vol. 324, p. 119624, Oct. 2022, doi: 10.1016/j.apenergy.2022.119624.

[57] R. van Dinter, B. Tekinerdogan, and C. Catal, "Predictive maintenance using digital twins: A systematic literature review," *Information and Software Technology*, vol. 151, p. 107008, Nov. 2022, doi: 10.1016/j.infsof.2022.107008.

[58] Z. Liu, Y. Chu, G. Li, H. P. Hildre, and H. Zhang, "Shipboard crane digital twin: An empirical study on R/V Gunnerus," *Ocean Engineering*, vol. 302, p. 117675, Jun. 2024, doi: 10.1016/j.oceaneng.2024.117675.

[59] E. Branlard, J. Jonkman, C. Brown, and J. Zhang, "A digital twin solution for floating offshore wind turbines validated using a full-scale prototype," *Wind Energy Science*, vol. 9, no. 1, pp. 1–24, Jan. 2024, doi: 10.5194/wes-9-1-2024.

[60] T. Kulik, Z. Kazemi, and P. G. Larsen, "Security and Privacy-related Issues in a Digital Twin Context," in *The Engineering of Digital Twins*, J. Fitzgerald, C. Gomes, and P. G. Larsen, Eds., Cham: Springer International Publishing, 2024, pp. 313–344. doi: 10.1007/978-3-031-66719-0_13.

[61] J. Fitzgerald *et al.*, "The Potential of Digital Twins: Four Industry Perspectives," in *The Engineering of Digital Twins*, J. Fitzgerald, C. Gomes, and P. G. Larsen, Eds., Cham: Springer International Publishing, 2024, pp. 19–43. doi: 10.1007/978-3-031-66719-0_2.