Chapter 2 – Twin, Shadow, Model, Simulation: What a Digital Twin Actually Is – and Isn't

2.0 Before you start

Where we are. Chapter 1 argued that a digital twin is bought to improve a decision, and it ran the whole argument on one deliberately loose working sentence:

> *Software that keeps a model of one specific physical system in step with that system's actual state, and uses it to answer questions about that system that measurement alone cannot answer.*

That sentence was enough to talk about money. It is not enough to write a statement of work, review a vendor's claim, or tell a customer what they are getting. This chapter replaces it.

What you are assumed to know. Everything Chapter 1 assumed – HTTP APIs, JSON, time-series storage, reading someone else's Python – plus three things from Chapter 1 itself. Each is recapped in one sentence where it is used, so you will not be stranded if the details have faded:

- The five **value patterns**: monitor, diagnose, predict, decide, certify-and-train.
- The **value metric**: one measurable quantity, with a baseline, that the twin is supposed to move.
- The **plant demonstrator** and its REST API on port 8099.

Still no control theory, no numerical simulation, no soil physics.

What this chapter covers. Why the definition is worth a chapter; the three ingredients of any twin; the data-flow test that separates a digital model from a digital shadow from a digital twin; the difference between a model, a simulation and a simulator; the other definitions you will meet in the wild and how to reconcile them; five things a digital twin is not; two properties the taxonomy misses that your project will care about; what changes in your obligations when you cross the shadow/twin boundary; and a full classification of the plant demonstrator at four stages of its life.

What this chapter deliberately does not cover. Architecture – Chapter 3. How models are built or solved – Part II. Whether a twin deserves to be believed – Chapter 7. How to implement any of this – Part III. Standards in any depth – Chapter 13, though two of them supply definitions in Sec. 2.5.

Learning objectives

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

1. **Classify** any proposed or existing system as a digital model, a digital shadow or a digital twin by tracing its data flows, and **justify** the classification from the direction and the automation of each flow.
2. **Distinguish** a model from a simulation from a simulator, and **identify** which of the three a colleague means when they say "the model".
3. **Apply** the classification to hard cases – the dashboard, the offline simulator, the human in the loop – and **defend** the answer against the obvious objection.

4. **Predict** what changes in a project's engineering obligations when it crosses from shadow to twin: what you may promise, what you must now verify, and which failure modes appear that did not exist before.
5. **Derive** the classification of the plant demonstrator at each stage of this book, and **specify** the exact change that moves it from shadow to twin.

2.1 Why the definition is worth a chapter

An argument about vocabulary is usually a waste of a meeting. This one is not, for four reasons that cost real money.

First, the term genuinely has no agreed meaning. This is not a rhetorical concession. A review of unit-level twin applications in manufacturing found the term used inconsistently across the literature it surveyed, with most references not sharing a definition, and named that lack of precision as the direct cause of confusion and misunderstanding [1]. A paper cataloguing twin workflows and architectures opens by stating flatly that there is no consensus on terminology [2]. A framework paper aimed at manufacturers lists the lack of consensus on the concept as a complicating factor for the small and medium enterprises trying to adopt it [3]. When a field's own reviews say this, treat "we all know what a digital twin is" as a warning sign.

Second, the disagreement has a shape, and the shape is useful. A survey of the asset-administration-shell literature observes that the definitions in circulation are almost as broad as the characteristics and use cases people claim, and then points at one definition that has found broad consensus – the one built on *level of integration* [4]. That is the one this chapter adopts, in Sec. 2.3, and the reason it has consensus is that it asks a question with a checkable answer.

Third, the confusion is expensive in exactly the way Chapter 1 warned about. Recall Chapter 1's market figures: forecasts for the same year differed by more than a factor of two. A category that can mean a dashboard or a coupled multi-physics simulation cannot be totalled. A standardisation study captured the same problem from the practitioner side, quoting a survey respondent's complaint that because there is no single definition, "many colleagues mean simulation when referring to DTs", and arguing that the standard ought to supply unified definitions of digital model, shadow and twin [5].

Fourth – and this is the one that will land on your desk – the boundary is contractual. "We will deliver a digital twin" is a sentence someone will sign. If your customer reads it as *the software will adjust the machine* and you built software that displays the machine, you have a dispute, and you will lose it. Being able to say, in the kickoff meeting, "what we are scoping is a digital shadow, and here is exactly what would make it a twin" converts a future argument into a present decision.

That sentence is the deliverable of this chapter. Everything below exists to let you say it and defend it.

2.2 Three ingredients

Strip away every disagreement and the same three ingredients survive.

1. **A Physical Twin (PT).** A *specific* real thing. Not a class of things – not "the AD20P pump" but *this* pump, on *this* bench, with the scale build-up it has. Chapter 1 pinned this term and it is unchanged.
2. **A digital object.** The software side: models, stored state, data and services that represent that one physical twin. This book uses the neutral phrase *digital object* on purpose, so that classifying a system stays a question about how data moves rather than about how clever the software is.
3. **A connection.** The data flowing between them, in one direction or both.

The three-part shape is the oldest thing in the field. Chapter 1 traced the concept to Grieves' 2002 presentation, whose model had exactly these parts: a real space, a virtual space, and the link between them [6]. Histories of the field report the same trio under slightly different labels, counting the physical entity and its digital counterpart as two of the parts and the link between them as the third [7].

Notice what is *not* on the list: three-dimensional geometry, machine learning, a cloud platform, a user interface. Every one of those appears in real twins. None of them is constitutive. Keep the list at three, because the third ingredient is where the whole chapter happens.

2.3 The data-flow test

Here is the definition this book uses from now on. It comes from a categorical literature review of digital twins in manufacturing, which observed that published definitions differ not in what they *describe* but in the level of data integration they assume, and separated the three cases by name [8].

2.3.1 The three cases

Trace the data. Ask about each direction separately, and ask whether the flow is *automated* or requires a person to carry it.

> **Digital model (DM).** A physical twin and a digital object both exist, and there is no automated data flow in either direction. A change in the physical object does not change the digital object, and a change in the digital object does not change the physical object. Any data that moves is moved by a person [8].
>
> **Digital shadow (DS).** There is an automated one-way data flow from the physical object to the digital object. A change in the state of the physical object leads to a change in the state of the digital object – but not the other way round [8].
>
> **Digital twin (DT).** The data flows are automated in *both* directions. The physical object changes the digital object, and the digital object changes the physical object [8].

The distinction has been picked up widely enough that you will meet it under several phrasings. One review describes the twin case as the digital object having an automated data flow back to the physical asset, so that changes on the digital object also entail changes on the asset [9]. Another states it as manual exchange in both directions being a model, automatic physical-to-digital without a reaction being a shadow, and bilateral communication being a twin [10]. An architecture-pattern catalogue frames the same three as design alternatives: no dataflow or manual only; sensors providing an automatic flow inward; and, in the third alternative, the digital and physical objects connected in both directions [11]. A reporting framework for twins uses the same axis and calls models and shadows the "downgraded versions" of a twin [12]. A reproducibility study restates the three as the standard subcategories by level of integration with the physical twin [13].

That much agreement across independent groups is why this is the definition worth memorising. Figure 2.1 is the whole distinction on one page: three pictures that differ only in how many arrows are automated.

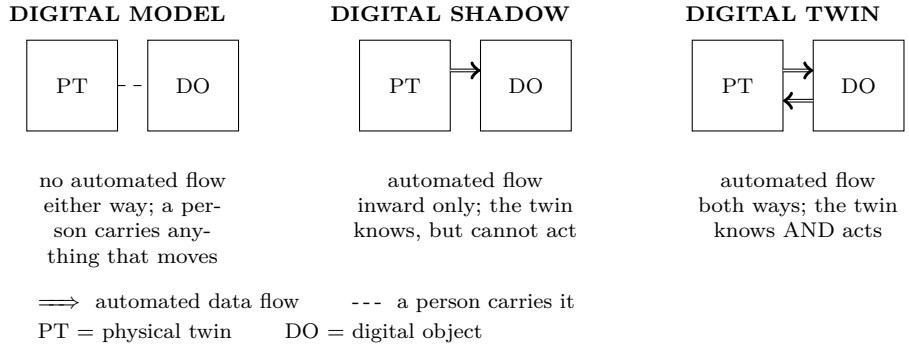


Figure 2.1 The data-flow test. The three classes differ in nothing except which arrows are automated – not in fidelity, not in cost, not in how impressive the visualisation is.

Note what the figure does *not* show, because that is the point of drawing it. There is no axis for model quality, no axis for how much data is collected, and no axis for how good the dashboard looks. A crude two-parameter model with both arrows automated is a twin; a photorealistic three-dimensional replica fed by a thousand sensors, with no way back to the physical twin, is a shadow.

2.3.2 The two-question test

Reduced to something you can run in a meeting:

```
Q1. Does the physical twin's state reach the digital object
    without a person carrying it?
      no  -> DIGITAL MODEL. Stop.
      yes -> go to Q2.

Q2. Does the digital object's output reach the physical twin
    and change what it does, without a person carrying it?
      no  -> DIGITAL SHADOW.
      yes -> DIGITAL TWIN.
```

Three properties of this test are worth naming, because each is a place people get it wrong.

It is about flows, not about sophistication. A digital object running a coupled multi-physics simulation, fed by hand from a spreadsheet once a month, is a digital model. A crude linear model fed automatically from a sensor and wired to a relay is a digital twin. The taxonomy is indifferent to how impressive the software is, which is exactly why it is useful – sophistication is the thing vendors compete on, and it tells you nothing about what the system will do.

"Without a person carrying it" is the load-bearing clause. Sec. 2.6.3 works through the case where a person is in the loop, because it is the question that comes up most and the one where the honest answer is the least comfortable.

It classifies a system, not a product. The same codebase can be a shadow in one deployment and a twin in another, if the second deployment enables an output path the first does not. Classify the deployment.

2.3.3 Why the test is drawn here and not somewhere else

A student is entitled to ask why *automated bidirectionality* is the line, rather than fidelity, or real-time-ness, or having a physics model. Three reasons, in increasing order of practical weight.

It is checkable. You can answer both questions by reading a network diagram and a deployment config. You cannot answer "is the model faithful enough?" that way – that takes Chapter 7 and a measurement campaign.

It is where the risk changes. A shadow that is wrong displays something false. A twin that is wrong *does* something false. That is a different class of failure, and Sec. 2.8 is about what it obliges you to do.

It is where the cost changes. Everything in the digital-to-physical direction – authorisation, command validation, rate limiting, interlocks, rollback, an audit trail, a safety case – is work a shadow never has to do. Crossing the line is not one more feature. It is a different project.

2.3.4 Replacing Chapter 1's working definition

As promised in Chapter 1, the loose sentence is now retired. The replacement:

> **Digital twin.** A digital object paired with a specific physical twin, where data flows automatically in *both* directions – physical to digital, and digital back to physical.

Chapter 1's sentence is not deleted, it is demoted. It described what a twin is *for*. The new one states what makes it a twin rather than a shadow. Both survive, at different jobs, and the book will use them that way: the Chapter 1 sentence when arguing value, the Chapter 2 sentence when classifying a system.

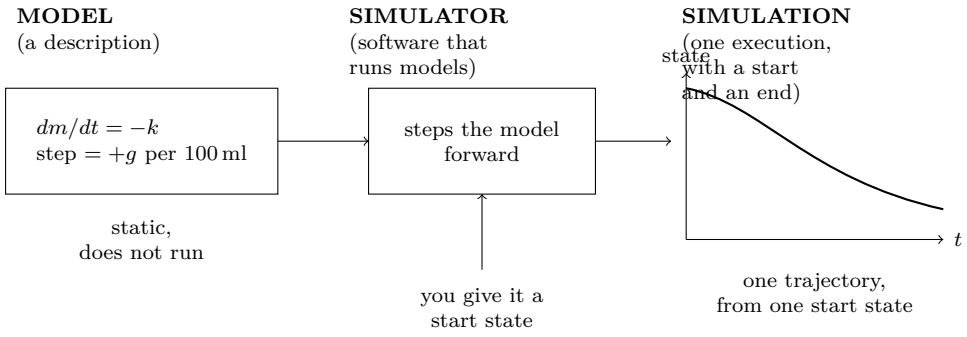
2.4 Model, simulation, simulator

Chapter 1 used the word "model" freely. Part II will use it constantly, and so will the modelling expert sitting across the table from you. Three distinct things share that word

in ordinary speech, and you will occasionally be the only person in the room who notices which one is meant.

> **Model.** A description of how something behaves – in equations, code, rules, a lookup table, or a fitted set of parameters. Static. It does not run.
>
> **Simulation.** One execution of a model over time, from a starting state, producing a trajectory of states. An event, not an object. You do not "have a simulation"; you *ran* one.
>
> **Simulator.** The software that performs simulations of a model.

Figure 2.2 shows how the three sit together: one is a thing, one is a process, one is a piece of software, and only the middle one has a beginning and an end.



You **HAVE** a model. You **HAVE** a simulator. You **RAN** a simulation.

Figure 2.2 The three words that share the word "model" in ordinary speech. The verb in the last line is the whole distinction.

The tenses in that last line are the test. If someone says "we have a simulation", ask which of the other two they mean – they have a simulator, or they have a model, or they ran something once and kept the output.

A concrete instance, from the demonstrator, so the three are not abstract.

- The **model**: "the soil's moisture reading drops by about `k` units per hour of daylight, and rises by about `g` units per 100 ml of water delivered." Two parameters, one sentence, no execution.
- A **simulation**: "starting from a reading of 620 at 06:00, with a 100 ml dose at 09:16, what is the reading at 18:00?" – run it and you get a curve. Change the start time and you have a *different simulation of the same model*.
- The **simulator**: the twenty lines of Python that step the model forward and hand back the curve. Chapter 6 is a survey of the kinds of simulator that exist, because "step it forward" hides an enormous amount of machinery once the model is not two parameters.

Why the distinction earns its place. Two failures follow directly from collapsing these words.

The first is the one the standardisation survey caught in the wild: colleagues who "mean simulation when referring to DTs" [5]. A simulation is a run. A twin is a standing pairing with a *specific* physical object. You can simulate an aircraft design that does not exist; you cannot twin it, because there is no particular aircraft for the data to flow from. **A**

twin needs a referent; a simulation does not. That single sentence resolves most arguments about whether a simulation project is "really" a twin project.

The second failure is subtler and it is a scheduling failure. "We already have a model" is often true and usually means the model exists as a document, a set of equations, or a report – not as something that can be executed on demand against live data. The distance between a model and a simulator you can call from a service every ten minutes is where a surprising number of twin projects lose their first quarter. A review of the move from modelling and simulation to digital twins lays out the workflow this hides – conceptual model, executable model, verification that the code implements the conceptual model, then calibration of its parameters – as distinct steps, each of which can fail on its own [14].

One more relationship, since it will come up. Models and data are complementary, not alternatives. Data from the physical twin tells the twin what state the physical twin is in; models tell it how the physical twin is expected to behave from that state [15]. A twin with data and no model can only report the present. A twin with a model and no data can only describe a generic object. It takes both.

2.5 The other definitions you will meet

You will be handed definitions that are not this chapter's. Here are the four you are most likely to meet, what each is optimised for, and how to reconcile it with the data-flow test. Reconciling rather than arguing is the professional move: these definitions mostly are not competing, they are answering different questions.

2.5.1 The aerospace one

The aerospace definition originates with the National Aeronautics and Space Administration (NASA), and is quoted across the literature in its own words: a twin is "an integrated multi-physics, multi-scale, probabilistic simulation of an as-built vehicle or system, using the best available physical models, sensor updates and fleet history to mirror the life of its flying twin" – an ultra-realistic, high-fidelity conception, as reproduced in [14].

What it is optimised for: a domain where being wrong kills people, and where the twin's job is life prediction on a specific airframe. Note "as built" and "its corresponding flying twin": the referent is specific, which is consistent with Sec. 2.4.

How to reconcile it: it is a definition of a *very high-fidelity* twin, not of the category. Read it as a description of one end of a spectrum. If you apply it as the entry requirement, almost nothing in industry qualifies, including systems that are unambiguously twins by the data-flow test and are paying for themselves.

2.5.2 ISO 23247's

The manufacturing standard itself defines a digital twin as a "fit-for-purpose digital representation of an observable manufacturing element, with a means to enable convergence between the element and its representation at an appropriate rate of synchronisation" [16]; the same wording is reproduced throughout the review literature [1].

What it is optimised for: being auditable and purpose-relative. Three phrases are doing work. **"Fit for purpose"** is Chapter 1's fidelity principle in standards language – adequacy is judged against the intended use, not in the abstract. **"Observable"** quietly rules out twinning what you cannot measure, which is Chapter 1's fourth stop condition. **"An appropriate rate of synchronisation"** introduces the property Sec. 2.7 takes up.

How to reconcile it: compatible with the data-flow test and more demanding in one respect – it asks not only whether the flows exist but whether they are fast enough for the purpose. Take it as a refinement, not a rival.

2.5.3 The consolidated academic one

A generalisation exercise across the published definitions proposes a twin as "a virtual representation of a physical system, and its associated environment and processes, that is updated through the exchange of information between the physical and virtual systems" [17]. That paper's own analysis of the field is worth knowing: it observes that definitions vary along representation fidelity, data collection and exchange, synchronisation frequency, and model or simulation capability, and it deliberately retains only what is essential, leaving out anything tied to a single use case.

How to reconcile it: "exchange of information" is the same axis as the data-flow test, stated without committing to a direction. It is the most compatible of the four and the least operational – it will not settle an argument about a dashboard, because it does not say which way the exchange has to go.

2.5.4 Grieves' three types

The oldest working taxonomy splits twins by lifecycle position rather than by integration. Its three categories sort by *when in a product's life the twin exists.* A **Digital Twin Prototype (DTP)** belongs to the design stage: nothing has been manufactured yet, and what the twin holds is whatever a factory would need in order to make one. A **Digital Twin Instance (DTI)** belongs to a single manufactured unit – it is tied to that one serial number and stays tied to it for as long as the unit exists. A **Digital Twin Aggregate (DTA)** is what you get by looking at many instances at once, and it is the only one of the three that can answer a question about a population rather than about a thing [6], [18]. Accounts of the concept's history place the three-subtype idea in the mid-2000s [7], and it is still in active use, including in teaching contexts [19].

How to reconcile it: this axis is **orthogonal** to the data-flow test, not in competition with it. A DTI can be a model, a shadow or a twin. The useful combination is to state both: "a digital twin instance, currently operating as a shadow." That sentence is precise, and it is the kind of sentence that ends a circular meeting.

2.5.5 The reconciliation table

Definition	Asks	Answers "is my dashboard a twin?"
Data-flow test (this book)	Which directions are automated?	Yes – and the answer is no

Definition	Asks	Answers "is my dashboard a twin?"
NASA	How faithful, on a specific as-built article?	Not directly; sets a high bar
ISO 23247	Fit for purpose, and synchronised fast enough?	Partly – adds a rate question
Consolidated academic	Is information exchanged?	No – direction unspecified
Grieves DTP/DTI/DTA	Where in the lifecycle?	No – different axis

The row that answers the question is the row this book uses. That is the only reason it was chosen over the others.

2.6 Five things a digital twin is not

Negative definitions teach faster than positive ones, because the reader arrives with the wrong ones already installed.

2.6.1 Not a 3D model

Geometry is a representation of *shape*. A twin is about *state and behaviour*. The two get conflated because twin marketing is visual and because a rendered plant floor photographs better than a residual plot.

Visualisation is genuinely valuable – it lets a user spot features, patterns, trends and relationships that are not apparent in raw data, and can surface issues that a model alone would obscure [20]. Platform comparisons note that twins present data as 3D models and interactive dashboards to help stakeholders follow complex processes [21]. Note the verb in both: it *presents*. Presentation is a service the twin offers, and Chapter 11 covers it properly. It is not the twin.

The tell: ask what happens if you delete the geometry. If the twin still answers its questions, the geometry was a view. If the whole thing stops, you had a Computer-Aided Design (CAD) model with a data feed.

2.6.2 Not a monitoring dashboard – and this is the promise Chapter 1 made

Chapter 1 said, of the *monitor* value pattern, that many systems sold as digital twins stop there, and that whether that counts as a twin at all is the argument this chapter settles. Settling it:

A monitoring dashboard is a digital shadow, not a digital twin.

Run the test. Q1: does the physical twin's state reach the digital object automatically? Yes – that is what the dashboard is. Q2: does the digital object's output change what the physical twin does, without a person carrying it? No. A dashboard's output goes to a human retina. **Digital shadow.** The taxonomy's own illustrative example of a shadow is exactly this: a car's dashboard representing properties of interest such as mileage and speed [9].

Three things follow, and the third is the one that matters commercially.

This is not an insult. Chapter 1's whole worked example – the fault detector with a nine-month payback – is a shadow. Shadows earn money. Reviews of the field consistently report finding more published work on digital models and digital shadows than on digital twins proper [8], which is a fair reflection of where value is actually being captured rather than a criticism of the field.

But you must not sell it as the other thing. The category difference is real, the customer's expectation attaches to the word, and the gap is exactly the work in Sec. 2.8.

And the honest sentence is short. "This is a digital shadow. It will tell you within eight hours that pot 7 stopped receiving water. It will not water pot 7. Making it water pot 7 is a separate increment, and here is what that costs." Nobody has ever been fired for that sentence.

2.6.3 Not necessarily a twin because a human acts on it

The hard case. A shadow detects the fault, raises an alert, a technician reads it and edits the watering schedule. The loop *is* closed – through a person. Twin or shadow?

By the data-flow test, shadow. The test says "without a person carrying it", and a person is carrying it.

That answer feels legalistic, so here is why the line is drawn where it is, which is the part worth learning. When a human is in the loop:

- A human applies judgement the digital object does not have, and rejects commands that are plainly wrong. Remove the human and you must build something that does that job.
- The system's response time is bounded by human availability – shift patterns, weekends, holidays. That is a design property, and it is usually the one being bought.
- Accountability rests with an identified person. Remove the human and accountability moves to your software and, in a real sense, to you.

Those three differences are not vocabulary. They are the difference between two systems with different failure modes, different response times, and different liability. The taxonomy tracks the important distinction; it just tracks it under a word that sounds pedantic.

That said, be precise rather than dogmatic. Practitioners recognise human-in-the-loop as its own twin *style*, distinct from a plain twin or shadow [22], and mobility twins have been demonstrated with a human driver deliberately kept in the loop [23]. The professional formulation is not "that is only a shadow" but: **"a digital shadow with a human-in-the-loop actuation path"** – which says exactly what exists, and makes the remaining gap to a twin visible.

2.6.4 Not a simulator

Sec. 2.4 covered the vocabulary; here is the operational tell. A simulator answers questions about a *class* of systems or a hypothetical one. A twin answers questions about *this* system, the one with the serial number, in the state it is in right now. A simulator with no referent and no live data is a digital model in the Sec. 2.3 sense, however good it is.

A crisp diagnostic: **ask which physical object would have to be destroyed for the software to become meaningless.** If there is no such object, you do not have a twin. Note that this is not a criticism of the simulator – it is usually the most valuable asset the project has, and it is the thing a twin gets *built around*.

2.6.5 Not a data lake, and not a machine-learning model

Storage is not a twin: a historian holding five years of sensor readings has no model, so it cannot answer a question about a state that was never measured. A trained network is not a twin either: it is a *model*, one candidate for the model slot inside a twin, and Chapter 8 covers what changes when you put a learned model there instead of a physical one.

The general form of all five negatives: each names a *component* or a *service* of a twin and mistakes it for the whole. Geometry, the dashboard, the simulator, the store and the learned model are all things twins contain. None of them is what makes the thing a twin. The connection is.

2.7 Two properties the taxonomy does not capture

The data-flow test gives you a category. Your project will need two more numbers that the category does not carry. Both come up in the first design review, and neither has a right answer independent of the value metric.

2.7.1 How often (twinning rate, age of twin)

ISO 23247's phrase "an appropriate rate of synchronisation" is pointing at this [1]. Twin platform surveys make it a measurable metric: the **twinning rate** relates how often the physical system issues updates to how often the digital twin takes them up, which is to say how often synchronisation actually happens; let it fall and the twin's state goes stale. The companion metric is the **age of twin**, the time elapsed since the digital state was refreshed [24].

The engineering consequence is that "real time" is not a requirement, it is an abdication. The useful requirement is derived from the decision, exactly as fidelity was in Chapter 1:

> How stale can the digital state be before the decision it feeds becomes wrong?

For the demonstrator's fault detector, the decision is "tell a technician that a dose did not land." Doses happen twice a day. A digital state one hour old is fine; one week old is useless. That reasoning gives a sampling period, and it gives it *without anyone saying "real time"*.

2.7.2 How late (time discrepancy)

Related and not the same. **Time discrepancy** is the gap between the moment the physical twin was in a state and the moment the digital object reflects it – lag along the path, rather than time since the last update. It has been studied in its own right for twins of cyber-physical systems, motivated by cases where the delay is the whole problem:

control loops destabilised by delay, or a twin warning a physical twin about an obstacle where any lag can make the warning too late to act on [25].

The two get conflated constantly, and Figure 2.4 separates them. Twinning rate is about the spacing of the updates; time discrepancy is about how far behind each one arrives.

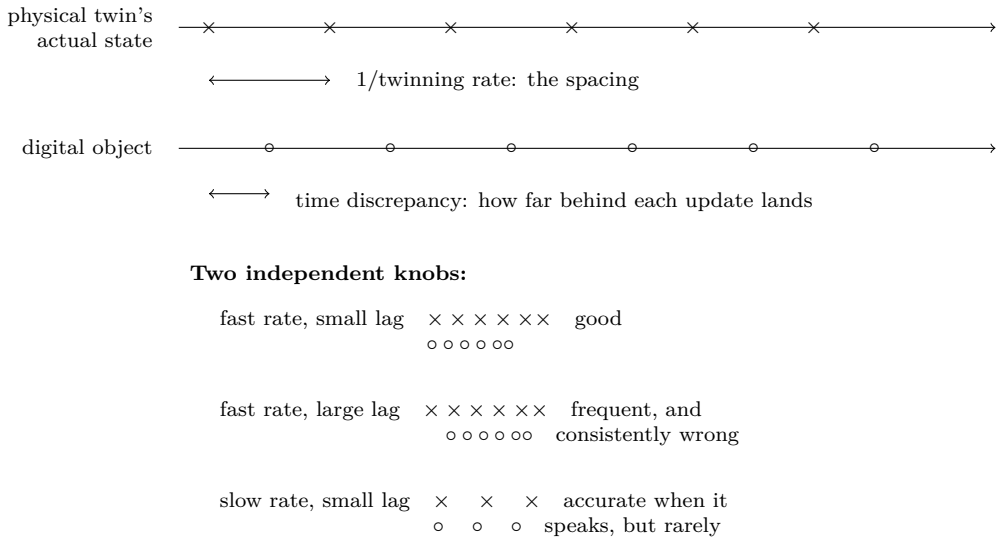


Figure 2.4 Twinning rate and time discrepancy are different failures. Raising the update rate does nothing for a pipeline that adds a fixed delay.

Both properties matter more the moment you cross into twin territory, which is the subject of the next section. A stale shadow shows an old number. A stale twin acts on an old number.

2.8 What changes when you close the loop

This is the "why the shadow distinction changes what you can promise" section Chapter 1 pointed at.

Closing the loop – adding the automated digital-to-physical flow – is the single act that turns a shadow into a twin. It is the difference that gives a twin its ability to support reconfiguration of the physical twin with some degree of autonomy, letting the physical twin respond to changes in its environment [26]. The framing in the cyber-physical-systems literature is careful about the order: first you build enough confidence to trust conclusions drawn on the twin, and only then can you envision the twin informing decisions that change the real world – and only where that confidence has actually been earned [27].

Here is the ledger of what changes. Read it as the scope difference between two quotes.

	Digital shadow	Digital twin
Worst case if the model is wrong	Displays something false	Does something false
Who is accountable for an action	The person who acted	Your software

	Digital shadow	Digital twin
Command path	None	Authorisation, validation, rate limits, interlocks
Reversibility	Not applicable	Must be designed for; some actions are irreversible
Staleness	Shows an old number	Acts on an old number (Sec. 2.7)
Testing	Against recorded data	Against the physical twin, or a trusted stand-in
Failure of the connection	Stale display; degrade gracefully	Must fail safe, which you have to define
Credibility evidence	Useful	Mandatory (Chapter 7)
Regulatory exposure	Usually low	Often the reason the project needs approval

Four consequences for how you work.

One: the safety question becomes yours. A shadow that reports a wrong moisture reading wastes a technician's walk. A twin that acts on it can flood a bench overnight. Before writing the command path, write down what the worst thing the twin can command is, and what physically prevents it. Frequently the right answer is a physical interlock rather than a software check – a smaller reservoir, or a hard limit in the controller firmware that the twin cannot raise.

Two: you now need a state you can trust, not just data you can read. A shadow can be sloppy about whether its state estimate is right, because a human reads the number and applies judgement. A twin cannot. Work on knowledge equivalence between a twin and its physical system exists exactly because "does the twin know enough to be trusted with this decision?" is a question you can now be required to answer [28]. Chapter 7 is where that gets discharged.

Three: the failure modes are new, not merely worse. A shadow has one interesting failure: it stops updating. A twin adds commanding on stale state, commanding on a diverged model, commanding while a person is also commanding, partial command delivery, and the loop oscillating because the twin's action changes the measurement that triggered it. None of these exists in a shadow. All of them are ordinary distributed-systems problems wearing physical consequences, which is good news – it is your home turf.

Four: the incremental path is the normal path, and it is not a failure of ambition. Model, then shadow, then twin is a sensible order, and each stage pays for the next. It is Chapter 1's lean argument restated in this chapter's vocabulary: deliver a rung that pays for itself, then climb.

2.9 Worked example: classifying the plant demonstrator

Chapter 1 built the value case for the demonstrator. This section classifies it – at four stages, with the reasoning at each step, using nothing but the two-question test.

Two facts from Chapter 1 that we lean on. The physical twin is the pot, plant, soil, pump, tubing and the Raspberry Pi controller. The controller exposes a REST API on port 8099 with, among others:

```
GET /sensing/{unit}/{parameter}          measurements, with limit + since_timestamp
GET /actuation/{unit}/watering_events    watering history
GET /actuation/{unit}/show_schedule      the schedule now in force
PUT /actuation/{unit}/update_schedule    replace the watering schedule
```

Hold that list in view. The classification boundary of this entire chapter runs between the third line and the fourth.

The pot-and-plant setting is a recognised exemplar for exactly this kind of reasoning – a documented greenhouse twin uses plants in pots, drying soil, and per-plant sensor streams as its running case [29], and a closely related mini-greenhouse setup with soil-moisture sensors and pumps appears in work on twin lifecycle management [30]. We are not inventing a toy.

2.9.1 Stage 0 – today, as shipped

What exists. The controller samples sensors, writes readings to a time-series database on the Pi, and fires the pump on a fixed daily schedule. Nothing else.

Classify it. The trap here is to call the controller's database a digital object and start answering Q1. It is not. Recall Sec. 2.2: the physical twin is *the pot, plant, soil, pump, tubing and the controller*. The controller's own logging is the physical twin keeping records about itself, the way a car's odometer is not a twin of the car.

Verdict: no digital object at all. Not a model, not a shadow, not a twin. This matters more than it sounds: a project that starts here and believes it is starting at "shadow" underestimates itself by an entire increment of data-engineering work.

2.9.2 Stage 1 – a model in a notebook

What exists. You export a few weeks of readings, fit the two parameters of the water-balance model from Sec. 2.4 in a notebook, and produce a chart showing predicted against measured moisture. You do this by hand, once.

Run the test. Q1: does the physical twin's state reach the digital object without a person carrying it? **No** – you carried it, by exporting a CSV.

Verdict: digital model. A useful one. It is where you discover whether a two-parameter model can even represent this pot, which is the cheapest thing to find out early and the most expensive to discover in Stage 2. Note that this stage is *not* free and *not* skippable, and note that its output is knowledge rather than software.

2.9.3 Stage 2 – the fault detector

What exists. A service that every ten minutes calls `GET /sensing/plant1/moisture` and `GET /actuation/plant1/watering_events`, steps the model forward, computes the *residual* – Chapter 1's term for the difference between what the model says should be happening and what the sensors say is happening – and raises an alert when a scheduled

dose produces no moisture step. This is precisely Chapter 1's recommended first increment – the *diagnose* value pattern, roughly nine-month payback.

Run the test. Q1: state reaches the digital object automatically? **Yes** – the polling loop, over HTTP, no human. Q2: does the digital object's output change what the physical twin does, automatically? **No** – the output is an alert to a technician.

Verdict: digital shadow.

Now sit with that for a moment, because it is the most important sentence in this chapter:

> **Chapter 1's recommended, payback-positive first increment is not a digital twin.**

That is not an embarrassment and it is not a bait-and-switch. It is the normal, correct shape of a first increment, and it is why Sec. 2.6.2 insisted that calling something a shadow is not an insult. It also means that if the project was sold as "we will build you a digital twin", Stage 2 does not deliver what was promised – while delivering exactly what was *valued*. Catching that mismatch in the kickoff meeting rather than the acceptance review is the practical payoff of this entire chapter.

If a technician then edits the schedule by hand, the loop is closed through a person: still a shadow, and Sec. 2.6.3 gives you the precise phrase – *a digital shadow with a human-in-the-loop actuation path.*

2.9.4 Stage 3 – closing the loop

What changes. One thing. The service stops sending an alert and starts calling:

```
PUT /actuation/plant1/update_schedule
```

Run the test. Q1: yes, unchanged. Q2: **yes** – the digital object's output now reaches the physical twin and changes what it does, with no person in the path.

Verdict: digital twin. The whole chapter, in one HTTP verb.

That is deliberately anticlimactic, and the anticlimax is the lesson: the categorical difference is a small change in the *data path* and a large change in *everything around it.* Walk Sec. 2.8's ledger against this one call and the size of the real change appears:

- **Worst case.** The model diverges, the twin concludes the pot is dry, and it writes a schedule with a large dose every hour. What physically stops it? Today, nothing – the reservoir volume, at best. A hard cap on total daily dose belongs in the *controller*, not in the twin, because the twin is the component whose correctness is in question.
- **Authorisation.** The endpoint takes a `PUT` from anything that can reach port 8099. That is acceptable for a shadow that never calls it. It is not acceptable once a program calls it unattended.
- **Staleness.** `since_timestamp` on the sensing endpoints lets the twin know how old its state is. A twin must refuse to command on state older than some bound. Deciding that bound is the Sec. 2.7 exercise, and the answer follows from the decision, not from the word "real-time".

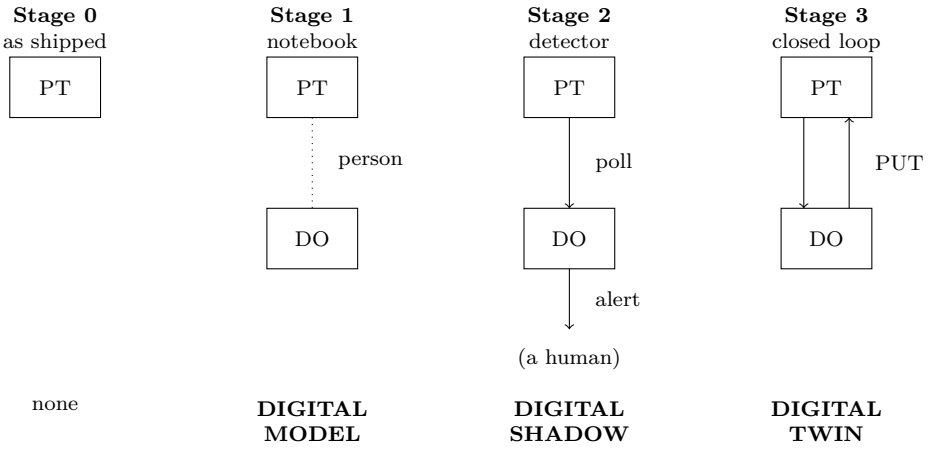
- **Concurrency.** `update_schedule` *replaces* the schedule. If a technician edits it while the twin also writes, the last writer wins silently. In a shadow this cannot happen. In a twin it is a data race with a plant at the end of it.
- **Reversibility.** A schedule can be rewritten. Water already pumped cannot be unpumped. Design the command path around the irreversible half.

Notice that none of those five is a modelling problem. All five are distributed-systems problems – authorisation, staleness, concurrency, idempotency, blast radius – which is why Part III is the deep part of this book for a reader with your background.

2.9.5 The classification table

Stage	What exists	Q1 auto in?	Q2 auto out?	Class
0	Controller only, fixed schedule	no digital object	–	none
1	Hand-fitted model in a notebook	no	no	Digital model
2	Polling fault detector, alerts a human	yes	no	Digital shadow
2h	As 2, technician edits schedule	yes	via human	Digital shadow, human in the loop
3	Detector writes the schedule via PUT	yes	yes	Digital twin

Figure 2.3 puts the same four stages on one picture, so the shape of the climb is visible: each stage adds exactly one arrow, and only the last one changes the class to *twin*.



Stage 2h sits between 2 and 3: the alert reaches a technician who edits the schedule by hand. The arrow outward exists, but a person is carrying it, so the class is still shadow.

Figure 2.3 The demonstrator's climb. One arrow per stage. The jump that matters is 2 to 3, and it is a jump in obligation, not in modelling effort.

Notice how little of that climb is modelling work. The model at Stage 3 is the same two-parameter water balance as at Stage 1; what changed is who is allowed to act on it, which is why Sec. 2.8's ledger of new obligations is the expensive part of the diagram rather than the box marked `DO`.

One last observation, and it is the demonstrator's own doing. Its documentation names two actors, and describes the second – the *Remote Connecter* – as "either a person or a computer running a Digital Twin". The physical twin was designed with the Stage 3 seam already cut. Recognising that seam in someone else's system, before you write any code, is the skill this chapter has been teaching.

2.10 Faded example: classify three systems

Same method, less scaffolding. For each, name the physical twin, the digital object, answer Q1 and Q2, and classify. I work the first, start the second, and leave the third to you.

System A – the hotel energy report. A hotel chain's engineering team receives a monthly PDF comparing each hotel's energy use against a building-physics model of that specific building. An analyst exports meter data, runs the model, and writes the report.

Worked. Physical twin: one specific hotel building. Digital object: the building-physics model plus the exported data. Q1: does meter data reach the model automatically? No – an analyst exports it. **Digital model.** Note that the model is sophisticated and building-specific, and it changes nothing: Sec. 2.3.2's first property.

System B – the offshore turbine anomaly service. Vibration and temperature stream from a turbine to shore continuously. A service compares them against a model of that turbine and raises a work order in the maintenance system when the residual exceeds a threshold. A planner schedules a vessel.

Started. Physical twin: turbine number 12. Digital object: the model plus the streaming state and the anomaly service. Q1: **yes**, the stream is automatic.

Q2 is yours. Careful – the work order *is* automatic, and it does reach a physical outcome eventually. Does it change what the turbine does without a person carrying it?

- **(a)** Classify System B and justify it in one sentence.
- **(b)** The vendor calls it a digital twin. Write the two-sentence reply you would give the customer that is accurate without being combative.
- **(c)** Name the single change that would make System B a twin under the test, and one thing that change would oblige the vendor to build that they do not need today.

Hint for (b): Sec. 2.6.3 supplies a phrase that describes what exists without conceding the label.

System C – the datacentre cooling controller. Rack inlet temperatures stream to a service every second. The service runs a thermal model, computes a set of fan speeds and chilled-water setpoints, and writes them to the building management system. Operators can override at any time but rarely do. Once a week the service retrains its thermal model on the last week of data.

Unaided. Classify it, then answer two harder questions: does the weekly retraining change the classification, and if the operators' override capability is exercised twice a year, does the system stop being a twin on those two days?

2.11 Posed problem: write the clause

Your company is about to sign a statement of work. The customer's draft says:

> "The Supplier shall deliver a Digital Twin of the Customer's bottling line, providing real-time visibility and enabling optimisation of line performance."

Rewrite that sentence as three numbered deliverables that are individually testable, using this chapter's vocabulary. At least one must be a digital shadow, at least one must state a synchronisation requirement derived from a decision rather than from the phrase "real-time", and at least one must make explicit whether anything is written back to the line.

Then write the two sentences you would say in the meeting when the customer asks why you replaced their sentence with three of yours.

2.12 Summary

Against the objectives this chapter opened with.

1. **Classification is a question about data flows.** No automated flow in either direction: digital model. Automated physical-to-digital only: digital shadow. Automated in both directions: digital twin [8]. Two questions, answerable from a network diagram (Sec. 2.3.2). It is indifferent to how sophisticated the software is, and that indifference is the feature.

2. **Model, simulation, simulator are three things.** A description; one execution of it; the software that executes it (Sec. 2.4). And the sentence that settles most arguments: a twin needs a referent, a simulation does not.
3. **The hard cases have answers.** A monitoring dashboard is a shadow, and that is not an insult (Sec. 2.6.2). A human closing the loop leaves it a shadow, best described as *a digital shadow with a human-in-the-loop actuation path* (Sec. 2.6.3). A simulator with no referent is a model (Sec. 2.6.4). The other definitions in circulation – NASA's, ISO 23247's, the consolidated academic one, Grieves' three types – mostly answer different questions and can be reconciled rather than fought (Sec. 2.5).
4. **Crossing the boundary changes the project, not the feature list.** A wrong shadow displays something false; a wrong twin does something false. Authorisation, staleness bounds, concurrency, reversibility, fail-safe behaviour and credibility evidence all arrive at once (Sec. 2.8), and the twinning rate and time-discrepancy questions get sharper teeth (Sec. 2.7).
5. **The demonstrator classifies cleanly, and the result is uncomfortable in a useful way.** Stage 0: no digital object. Stage 1: digital model. Stage 2 – Chapter 1's payback-positive fault detector – digital *shadow*. Stage 3: one `PUT /actuation/{unit}/update_schedule` makes it a digital twin, and that single HTTP call drags five distributed-systems obligations in behind it (Sec. 2.9).

The sentence to carry forward, and to say out loud in your next kickoff meeting: **"What we are scoping is a digital shadow, and here is exactly what would make it a twin."**

2.13 Exercises

Objectives in brackets. Solutions and hints follow.

2.13.1 (Objective 1, easy). Classify each, in one line, with the flow that decides it: (a) a car's dashboard showing speed and mileage; (b) a finite-element model of a bridge, run once during design; (c) a thermostat that reads a room sensor and drives a boiler; (d) a weather forecast.

2.13.2 (Objective 2, easy). A colleague says "we already have the model, so the twin is mostly done." Name three distinct things that sentence might mean, and the question you would ask to find out which.

2.13.3 (Objective 1, easy). Chapter 1's five value patterns were monitor, diagnose, predict, decide, certify-and-train. For each, state the *minimum* class from this chapter that can deliver it, and say which pattern is the first that requires a twin rather than a shadow.

2.13.4 (Objective 3, medium). Complete System B in Sec. 2.10, parts (a) through (c).

2.13.5 (Objective 3, medium). Do System C in Sec. 2.10, including both harder questions.

2.13.6 (Objective 4, medium). For the demonstrator at Stage 3, write the staleness rule: the maximum age of the moisture state at which the twin is still allowed to write a

schedule, and the arithmetic that justifies it. State what the twin should do when the state is older than that.

2.13.7 (Objective 4, medium). Sec. 2.9.4 lists a data race: `update_schedule` replaces the schedule, so a technician and the twin can overwrite each other. Propose two fixes – one purely in the twin, one requiring a change to the controller's API – and say which you would advocate and why.

2.13.8 (Objectives 1 and 4, hard). Take a system you have personally built or operated that talks to hardware. Classify it with the two-question test. Then answer: if it is a shadow, what exactly would closing the loop oblige you to add? If it is a twin, which of Sec. 2.8's obligations did the project actually discharge, and which did it skip?

2.13.9 (Objective 3, hard). Sec. 2.5 claims the four rival definitions "mostly are not competing". Argue the opposite: construct a concrete system that one of the four would call a digital twin and the data-flow test would not, and say which definition you would use in a contract and why.

2.13.10 (Objectives 1 and 4, hard, open-ended). Do Sec. 2.11, the statement-of-work clause, in full.

Solutions and hints

2.13.1. (a) Digital shadow – automatic in, nothing automatic out; this is the taxonomy's own worked example [9]. (b) Digital model – no automated flow, and note it is also the most technically demanding item on the list, which is Sec. 2.3.2's first property again. (c) Digital twin, formally: sensor in, boiler command out, both automatic, one specific room. If that feels wrong, the discomfort is informative – the data-flow test is about structure, not scale, and a thermostat has the structure. What it lacks is a model worth the name, which is a *fidelity* question, not a classification one. (d) Neither – there is no specific physical twin. Earth is not a serial number. (A twin of one specific building or one specific field is a different matter.)

2.13.2. *Hint:* Sec. 2.4. The three meanings are roughly: a set of equations or a report (a description); an executable artefact that runs on demand; and a *calibrated* executable artefact whose parameters have been fitted to this specific physical twin. The question to ask: "can I call it with today's sensor readings and get an answer back in under a second, for *this* pot?" The workflow those three collapse is the one laid out in the modelling-and-simulation-to-twin literature – conceptual model, executable model, verification, calibration [14].

2.13.3. Monitor: shadow. Diagnose: shadow. Predict: shadow – a forecast is still output to a human. Decide: shadow *if* it advises, twin if it acts; this is the pattern where the boundary lives, and the answer depends on the deployment, not the pattern. Certify-and-train: often neither, since the physical twin may not exist yet – a training simulator with no referent is a digital model by Sec. 2.6.4. The first pattern that *requires* a twin is "decide", and only in its acting form. Worth noticing: four of Chapter 1's five patterns are deliverable by a shadow.

2.13.4. (a) **Digital shadow.** The work order is automatic, but it reaches a *planner*, not the turbine; nothing changes what the turbine does without a person. (b) *Hint:* use Sec. 2.6.3's construction – "a digital shadow with a human-in-the-loop actuation

path" – then say what it does deliver, which is genuine and valuable. Do not open by disputing the label. (c) The change: the service commands the turbine directly, for example derating or stopping it. New obligations include, at minimum, an authorisation path, a staleness bound, and a fail-safe behaviour when the link to shore drops – none of which a work-order-raising service needs, because a planner supplies all three implicitly.

2.13.5. *Partial.* **Digital twin** – setpoints are written automatically to the building management system. On the harder questions: (i) weekly retraining does not change the classification, because the taxonomy is about data flows between physical and digital, not about how the model was obtained; it does change Chapter 7's problem substantially, since a model that changes weekly must be re-validated weekly. (ii) No. The classification describes the system's designed data paths, not each day's events. An override is the human-authority mechanism Sec. 2.8 says a twin needs; exercising it is the system working as designed, not the system changing category.

2.13.6. *Hint:* the reasoning template is Sec. 2.7.1. Doses occur twice daily, and the quantity the twin acts on is soil moisture, which changes on a scale of hours. A defensible answer bounds staleness well inside one watering interval – an hour or two – and justifies it from the interval, not from a preference. The second half is the more important half: on exceeding the bound, the twin must *not* write, must leave the last known good schedule in force, and must raise an alert. "Fail to the last safe state and tell someone" is the general form.

2.13.7. *Hint:* in the twin, read `show_schedule` immediately before writing and abort if it is not what the twin last wrote – a compare-and-swap done badly, since it still races. In the API, add a version or ETag to the schedule and reject a `PUT` carrying a stale one. Advocate the API change: the race is in the resource, and a fix that lives only in one client stops working the moment there are two. That argument – put the invariant where the state is – is the same one you would make about any shared resource, which is the point of Sec. 2.9.4's closing observation.

2.13.8. Open. Assessed on whether the classification is justified from flows rather than from what the system *felt* like, and on whether the second half names obligations from Sec. 2.8 specifically rather than gesturing at "more testing".

2.13.9. *Hint:* the easiest construction uses the consolidated academic definition [17], which requires only that information be exchanged and does not name a direction – so a rich monitoring platform satisfies it and fails the data-flow test. A good answer then picks the data-flow test for the contract and says why: it is the one whose answer can be checked from a deployment diagram, and a contractual term that cannot be checked is not a term. A very good answer concedes the cost of that choice – the test says nothing about whether the twin is any good, which is why the contract also needs Chapter 7's clauses.

2.13.10. *Hint:* no solution, but a test. Read your three deliverables and ask, for each: could two engineers disagree about whether it has been met? If yes, it is still a sentence, not a deliverable. And check that one of the three explicitly says whether anything is written back to the line – that clause is the whole chapter.

2.14 Where to go next

In this book. Chapter 3 takes the three ingredients of Sec. 2.2 and turns the digital object into a reference architecture with named components, so that "the digital object" stops being one box. Part II gives you the modelling and simulation literacy that Sec. 2.4 only named – Chapter 6 in particular is the survey of simulators. Chapter 7 discharges the credibility obligation that Sec. 2.8 says arrives the moment you close the loop. Chapter 11 covers the visualisation and monitoring services that Sec. 2.6.1 and Sec. 2.6.2 declined to call twins. Chapter 13 owns the standards that supplied two of Sec. 2.5's definitions. Part III builds the demonstrator's Stage 2 and Stage 3.

In the literature. Grouped by what each is good for:

- *The taxonomy itself, first-hand:* [8] is the source of the model/shadow/twin split and the paper to read if you read only one. For independent restatements that each add a nuance, see [9], [10], [11], [12] and [13].
- *On there being no consensus, and why:* [2], [1], [3], [31] and [4], with [5] for the practitioner-survey view and the argument that standards should fix it.
- *Definitions from outside the taxonomy:* [14] for NASA's and for the modelling-and-simulation workflow of Sec. 2.4, [17] for the consolidation exercise, [6] with [18], [7], [19] and [32] for the prototype/instance/aggregate axis.
- *Synchronisation, staleness and lag:* [24] for twinning rate and age of twin as measurable metrics, [25] for time discrepancy as a problem in its own right, [33] for state synchronisation, and [34] for fidelity, synchronisation and integration treated together.
- *Closing the loop:* [26] on autonomous reconfiguration as the thing a twin can do that a shadow cannot, [27] and [15] for the book-length treatment this chapter's framing is closest to, and [28] on what a twin must know before it is trusted to act.
- *Classification in practice, on real systems:* [35] describes four twins against a fourteen-characteristic framework and is the best available answer to "what do real ones actually look like"; [29] and [30] are the closest published relatives of our demonstrator; [36], [37], [38] and [39] are worked examples in beer fermentation, shipboard cranes, operating rooms and hospital wards respectively.
- *Consulted, not drawn on above:* [20] on visualisation as a service (Chapter 11), [21] on platforms (Chapter 12), [22] on human-in-the-loop as a named twin style, [23] on mobility twins with a human driver, [40] and [41] on how other authors pin the same definitions, [42] and [43] on keeping models current, [44] on domain-specific readings of the term, [45] and [46] on composing twins (Chapter 15), [47] on adoption barriers, and [48] on the standards landscape (Chapter 13).

In the demonstrator. Open `pt/controller_3/src/plant_controller/web_api.py` and find the four endpoints listed in Sec. 2.9. Then find the one `PUT` among them. That single route is the boundary this chapter is about, and everything Part III builds sits on one side of it or the other.

2.15 References

[1] T. Böttjer *et al.*, "A review of unit level digital twin applications in the manufacturing industry," *CIRP Journal of Manufacturing Science and Technology*, vol. 45, pp. 162–189, Oct. 2023, doi: 10.1016/j.cirpj.2023.06.011.

[2] R. Paredis, C. Gomes, and H. Vangheluwe, "A Family of Digital T Workflows and Architectures: Exploring Two Cases," in *Innovative Intelligent Industrial Production and Logistics*, A. Smirnov, H. Panetto, and K. Madani, Eds., Cham: Springer Nature Switzerland, 2023, pp. 93–109. doi: 10.1007/978-3-031-37228-5_6.

[3] G. Shao and M. Helu, "Framework for a digital twin in manufacturing: Scope and requirements," *Manufacturing Letters*, vol. 24, pp. 105–107, Apr. 2020, doi: 10.1016/j.mfglet.2020.04.004.

[4] C. Ellwein *et al.*, "Rethinking Asset Administration Shell Communication Types: A Systematic Mapping Study and Portfolio-Based Classification," *Production Engineering*, vol. 20, Dec. 2025, doi: 10.1007/s11740-025-01378-3.

[5] 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.

[6] 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.

[7] F. Tao, M. Zhang, and A. Y. C. Nee, Eds., "Digital Twin Driven Smart Manufacturing," in *Digital Twin Driven Smart Manufacturing*, Academic Press, 2019, pp. i–iii. doi: 10.1016/B978-0-12-817630-6.00013-8.

[8] W. Kritzinger, M. Karner, G. Traar, J. Henjes, and W. Sihn, "Digital Twin in manufacturing: A categorical literature review and classification," *Ifac-PapersOnline*, vol. 51, no. 11, pp. 1016–1022, 2018, Accessed: Dec. 11, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896318316021

[9] J. Pfeiffer *et al.*, "Towards a Unifying Reference Model for Digital Twins of Cyber-Physical Systems." arXiv, Jul. 2025. doi: 10.48550/arXiv.2507.04871.

[10] T. L. Leirmo, "Digital Twins for Industry 5.0: Unlocking the Human Potential," *Procedia CIRP*, vol. 130, pp. 761–766, Jan. 2024, doi: 10.1016/j.procir.2024.10.161.

[11] 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.

[12] S. Gil, B. Oakes, C. Gomes, M. Frasheri, and P. Larsen, "Toward a systematic reporting framework for Digital Twins: A cooperative robotics case study," *SIMULATION*, Aug. 2024, doi: 10.1177/00375497241261406.

[13] A. Barbie and W. Hasselbring, "Toward Reproducibility of Digital Twin Research: Exemplified with the PiCar-X." arXiv, Aug. 2024. doi: 10.48550/arXiv.2408.13866.

[14] Z. Ali, R. Biglari, J. Denil, J. Mertens, M. Poursoltan, and M. K. Traoré, "From modeling and simulation to Digital Twin: Evolution or revolution?" *SIMULATION*, vol. 100, no. 7, pp. 751–769, Jul. 2024, doi: 10.1177/00375497241234680.

[15] 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.

[16] "ISO 23247-1:2021," *ISO*. Accessed: Mar. 28, 2025. [Online]. Available: https://www.iso.org/standard/75066.html

[17] E. VanDerHorn and S. Mahadevan, "Digital Twin: Generalization, characterization and implementation," *Decision Support Systems*, vol. 145, p. 113524, Jun. 2021, doi: 10.1016/j.dss.2021.113524.

[18] 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.

[19] M. Grieves, "Digital Twins and Their Role in Reengineering Engineering Education," in *Digital Twin: Fundamentals and Applications*, S. Sabri, K. Alexandridis, and N. Lee, Eds., Cham: Springer Nature Switzerland, 2024, pp. 237–261. doi: 10.1007/978-3-031-67778-6_11.

[20] C. H. Bohlbro, H. D. Macedo, D. Tola, L. Esterle, and P. G. Larsen, "Visualisation 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. 175–188. doi: 10.1007/978-3-031-66719-0_8.

[21] D. Parle, G. Sharma, N. Anand, N. Padgaonkar, D. Stoddart, and D. Malley, "A Comparative Analysis for Harnessing Digital Twin Platforms for Net-Zero Manufacturing," *IEEE Access*, vol. PP, Aug. 2024, doi: 10.1109/ACCESS.2024.3447475.

[22] X. Liu and I. David, "AI simulation by digital twins: Systematic survey, reference framework, and mapping to a standardized architecture," *Software and Systems Modeling*, Aug. 2025, doi: 10.1007/s10270-025-01306-0.

[23] Z. Wang *et al.*, "Mobility digital twin: Concept, architecture, case study, and future challenges," *IEEE Internet of Things Journal*, vol. 9, no. 18, pp. 17452–17467, 2022, Accessed: Jan. 08, 2025. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9724183/

[24] K. Duran *et al.*, "Toward Digital Twin-as-a-Service (DTaaS) Platforms: A Survey on Architecture, Design Requirements, and Performance Metrics," *IEEE Communications Surveys & Tutorials*, vol. 28, pp. 1845–1878, 2026, doi: 10.1109/COMST.2025.3635582.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] M. Liu, S. Fang, H. Dong, and C. Xu, "Review of digital twin about concepts, technologies, and industrial applications," *Journal of Manufacturing Systems*, vol. 58, pp. 346–361, Jan. 2021, doi: 10.1016/j.jmsy.2020.06.017.

[32] S. Sabri, K. Alexandridis, and N. Lee, Eds., *Digital Twin: Fundamentals and Applications*. Cham: Springer Nature Switzerland, 2024. doi: 10.1007/978-3-031-67778-6.

[33] H. Wu, P. Ji, H. Ma, and L. Xing, "A Comprehensive Review of Digital Twin from the Perspective of Total Process: Data, Models, Networks and Applications," *Sensors*, vol. 23, p. 8306, Oct. 2023, doi: 10.3390/s23198306.

[34] 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.

[35] 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.

[36] P.-E. Goffi, R. Tremblay, and B. Oakes, "Engineering a Digital Twin for the Monitoring and Control of Beer Fermentation Sampling." arXiv, Aug. 2025. doi: 10.48550/arXiv.2508.18452.

[37] 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.

[38] S. Burattini *et al.*, "An Ecosystem of Digital Twins for Operating Room Management," in *2023 IEEE 36th International Symposium on Computer-Based Medical Systems (CBMS)*, Jun. 2023, pp. 770–775. doi: 10.1109/CBMS58004.2023.00317.

[39] R. Sieve, P. Kobialka, L. Slaughter, R. Schlatte, E. B. Johnsen, and S. L. T. Tarifa, "BedreFlyt: Improving Patient Flows through Hospital Wards with Digital Twins," *Electronic Proceedings in Theoretical Computer Science*, vol. 418, pp. 1–15, May 2025, doi: 10.4204/EPTCS.418.1.

[40] Z. Chen *et al.*, "Service oriented digital twin for additive manufacturing process," *Journal of Manufacturing Systems*, vol. 74, pp. 762–776, Jun. 2024, doi: 10.1016/j.jmsy.2024.04.015.

[41] R. Delhibabu, A. Vodyaho, D. Ignatov, N. Zhukova, and M. Chervoncev, *Synthesis of multi-stakeholder run time dynamic digital twins and dynamic digital twins networks*. 2023. doi: 10.21203/rs.3.rs-3000825/v1.

[42] M. Heithoff, N. Jansen, J. Michael, F. Rademacher, and B. Rumpe, "Model-Based Engineering of Multi-Purpose Digital Twins in Manufacturing," in *Digital Twin: Fundamentals and Applications*, S. Sabri, K. Alexandridis, and N. Lee, Eds., Cham: Springer Nature Switzerland, 2024, pp. 89–126. doi: 10.1007/978-3-031-67778-6_5.

[43] 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.

[44] R. Richstein and K.-U. Schröder, "Characterizing the Digital Twin in Structural Mechanics," *Designs*, vol. 8, no. 1, p. 8, Feb. 2024, doi: 10.3390/designs8010008.

[45] M. S. Gill, J. Zhang, A. Wortmann, and A. Fay, "Toward Automating the Composition of Digital Twins Within System-of-Systems," in *2024 IEEE 29th International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sep. 2024, pp. 1–4. doi: 10.1109/ETFA61755.2024.10710740.

[46] A. Barbone, S. Burattini, M. Martinelli, M. Picone, A. Ricci, and A. Virdis, "Digital Twin Continuum: A Key Enabler for Pervasive Cyber-Physical Environments," in *2024 33rd International Conference on Computer Communications and Networks (ICCCN)*, Jul. 2024, pp. 1–9. doi: 10.1109/ICCCN61486.2024.10637565.

[47] M. Perno, L. Hvam, and A. Haug, "Implementation of digital twins in the process industry: A systematic literature review of enablers and barriers," *Computers in Industry*, vol. 134, p. 103558, Jan. 2022, doi: 10.1016/j.compind.2021.103558.

[48] "Summary of IoT, and DT Standards." Change2Twin project.