# Chapter 3: Model, Simulation, Shadow and Twin

## Before You Start

**What you are assumed to know.** You write software for a living, you are comfortable with HTTP APIs, JSON, a time-series or relational database, and reading Python you did not write, and you have seen a system that talks to hardware. You have read the previous two chapters, and this chapter leans on five things they established. From Chapter 1: that a **digital twin (DT)** is bought to improve one decision about one specific real system, the **physical twin (PT)**; that value comes in six patterns – *monitor, diagnose, predict, decide, control, certify* – each saying who acts on the twin's output; and that the **residual**, the gap between what a model expects and what a sensor reports, is the number most first increments are built around. From Chapter 2: the plant rig – a potted plant, a pump, and a Raspberry Pi controller that exposes everything through a REST API on port 8099 with seven endpoints that read and exactly one, `PUT /actuation/{unit}/update_schedule`, that writes – and the small mathematical model of both sides of that boundary: the pot dries at a rate $d$, a dose of $V$ millilitres raises the reading by $g$ per millilitre, and the twin predicts the next reading $\hat{m}$, compares it with the observed $m_{\text{obs}}$, and calls the difference $r$. Each of these is recalled in one clause where it is used, so a faded detail will not strand you.

**What you are not assumed to know.** Anything about control theory, numerical simulation, or the digital-twin literature. Where this chapter needs a term from one of those fields it defines the term at first use and then keeps using it.

**What this chapter covers.** The previous two chapters used the words *model*, *simulation* and *twin* freely, on one loose working definition, because their job was to say what a twin is *for* and to put a real rig on the desk. This chapter tightens the words. It separates a model from a simulation from a simulator; it gives the one test this book uses from now on to tell a **digital model** from a **digital shadow** from a **digital twin**, and shows that the test is about how data moves and about nothing else; it reads the returning arrow of that test as what it is, a control loop, and lists what closing it obliges you to build; it widens the referent from machines to software systems, processes and organisations; it sorts twins by what they twin – product, process, lifecycle, biological, ecological, sociological – and shows that the sorting is orthogonal to the test; it names five things a digital twin is not; and it classifies the plant rig at four stages of its life, ending on the single HTTP call that changes its class.

**What this chapter deliberately does not cover.** What is *inside* the digital object – its architecture, its components, how its model is built, solved or validated – which the chapters that follow take up. Standards in any depth: one standard supplies one definition here and nothing more. How often a twin should synchronise with its PT and how late its picture may run: both matter, both get one row in a table here, and both belong to the data chapters.

**Learning objectives.** By the end of this chapter you will be able to:

1. **Distinguish** a model from a simulation from a simulator, and **identify** which of the three a colleague means when they say "the model".
2. **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.
3. **Apply** the classification to the hard cases – the dashboard, the offline simulator, the human who closes the loop, the referent that is not a machine – 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, and **name** the checks a command path must make before it acts.
5. **Derive** the classification of the plant rig at each stage of its life, and **specify** the one change that moves it from shadow to twin.

Every section that follows serves at least one of those five, and the exercises at the end say which.

## Model, Simulation and Simulator

Three different things share the word "model" in ordinary speech, and in a project meeting you may be the only person who notices which one is meant. Pin them first, because the classification test in the next section depends on one of them and is defeated by confusing the other two.

> **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 1 puts the three side by side. One is a thing you can print, one is a program, and one is an event with a beginning and an end. The verbs on the bottom line are the test: if a colleague 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.

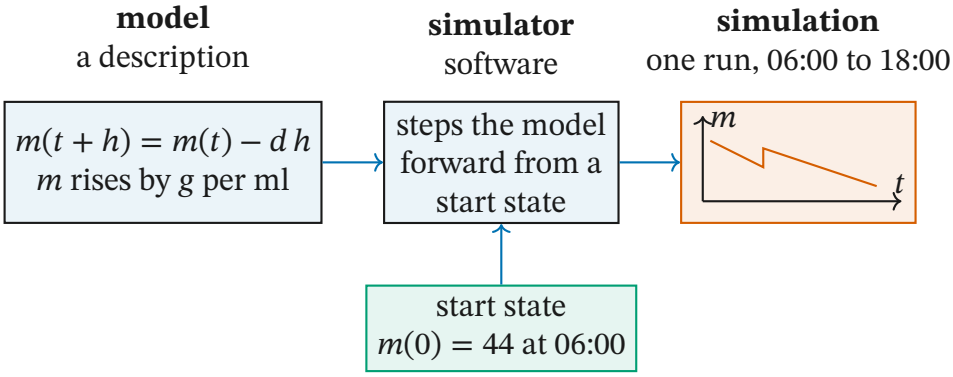



Figure 1: Three things that share one word. A model is fed, with a start state, into a simulator, and one run of it is a simulation. Only the last has a beginning and an end.

### The rig's own instance

The previous chapter wrote down two models, and they make the distinction concrete. Both are recalled here rather than re-derived.

The **model of the physical twin** describes the pot, the pump and the controller below the API. Two of its four equations matter here. With no water, the moisture reading falls at the drying rate:

$$m(t + h) = m(t) - d\,h$$

and a delivered dose raises it in proportion to the dose response:

$$m_{\text{after}} = m_{\text{before}} + g\,V_{\text{del}}$$

Two parameters, $d$ in percentage points per hour and $g$ in percentage points per millilitre, and two lines. Nothing executes. That is a model.

The **model of the digital twin** describes what the software above the API believes and does. It reads a moisture $m$, asks for a dose $V$ by the dose rule, and predicts the reading at the next check $T_{\text{c}}$ hours later:

$$\hat{m} = m + g\,V - d\,T_{\text{c}}$$

then, when that reading $m_{\text{obs}}$ arrives, computes the residual $r = \hat{m} - m_{\text{obs}}$. Also a model, and also static: three equations on a page, none of which has been run.

A note on the words, because the next section is about to reuse one of them. "The model of the digital twin" means *the twin's own equations*. "A digital model", two pages from now, will mean *a class of system* – a PT and some software with no automated link between them. The two phrases share a word and mean different things, and the chapter keeps them apart by always saying "the model **of** the digital twin" for the equations and "**a** digital model" for the class.

A **simulation** is one run of either model. Take the pot's model, start at $m(0) = 44$ at 06:00, assume the drying rate the previous chapter fitted, $d = 1.0$, and a dose response of $g = 0.1$ – a value this chapter assumes, since the previous one named $g$ and did not fit it – and deliver one dose of $V = 100$ millilitres at 09:16. What is the reading at 18:00? Twelve hours of drying take off twelve points; the dose puts back ten:

$$m(12) = 44 - 1.0 \times 12 + 0.1 \times 100 = 42$$

That arithmetic, and the curve it traces on the way, is a simulation. Change the start reading, or move the dose to 07:00, and you have a *different simulation of the same model.* Change $d$ and you have a different model.

The **simulator** is the dozen lines of Python that step the model forward hour by hour and hand back the curve. For a two-parameter model those dozen lines are the whole of it. Once a model has stiff dynamics, several coupled parts, or a physics library behind it, "step it forward" hides an enormous amount of machinery, and a later chapter is a survey of the kinds of simulator that exist.

### Why the distinction earns its place

Four things go wrong when the three words collapse into one, and each has cost somebody a quarter.

**People hear "twin" and picture a simulation.** A survey of practitioners on standardising twins reports two-thirds of respondents agreeing that a system with no interaction running back to the physical side should not be called a twin – and one respondent explaining the disagreement in the field by saying that, because there is no single definition, many colleagues mean simulation when they refer to digital twins [1]. A simulation is a run. A twin, as the next section defines it, is a standing pairing with one 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 one sentence settles most arguments about whether a simulation project is "really" a twin project.

**"We already have a model" is true and means less than it sounds.** The modelling-and-simulation literature separates the steps that phrase hides: a conceptual model in the developer's head, an executable or programmed model that captures it, *verification* that the program implements the concept without error, and then *calibration* of its parameters against the real system [2]. A model that exists as a report has done the first step. A model that a service can call every ten minutes against live readings from *this* pot has done all four. The distance between them is where a surprising number of twin projects lose their first quarter, and the question that measures it is "can I call it, with today's readings, and get an answer back for this plant?"

**A simulator is not a twin, however good it is.** The engineering-of-digital-twins literature puts it bluntly: having models, and running them, gets you no twin by itself, and a model earns its place in one only when readings from the PT can be fed into it; a hardware-in-the-loop rig that never feeds its outputs back to a deployed PT is not one either, and a *tracking simulator* – one kept in step with live data – is closest to what the next section will call a shadow [3]. The standards analysts say the same thing from the other side: simulations and twins both use models to replicate a product or a process, and the most notable difference is that a twin is fed by live measurements, kept in step with the thing measured, and can talk back to the sensors that supply them, which is what separates it from conventional offline modelling and simulation [4]. Both are pointing at the same fact. The difference between a simulator and a twin is not in the model. It is in the arrows.

**Models and data are complementary, not rivals.** The data from the PT tells the twin what state the PT is *in*; the model tells it how the PT is *expected* to behave from that state, and the twin's picture is refreshed whenever the real thing moves on, which is what lets anyone trust conclusions drawn on it [5]. The rig's residual $r$ is exactly that complementarity in one number: a prediction the model made, minus a reading the data supplied. 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 pot. It takes both, and the next section is about the arrows that carry the data half.

## Digital Model to Digital Twin

An argument about vocabulary is usually a waste of a meeting. This one is not, for two reasons that cost money. First, the term genuinely has no settled meaning: a manufacturing study of the two terms opens by admitting that nobody yet agrees what they cover, and that the disagreement gets in the way of building anything with them [6]; the practitioner survey quoted above found a two-thirds majority for "no bidirectional interaction, no twin", which is a majority and not a consensus [1]. Second, the boundary is contractual. "We will deliver a digital twin" is a sentence someone will sign. If the customer reads it as *the software will adjust the machine* and you built software that displays the machine, you have a dispute. Being able to say, at the kickoff, "what we are scoping is a digital shadow, and here is exactly what would make it a twin" turns that future argument into a present decision. This section exists so that you can say that sentence and defend it.

### Three ingredients

Strip away every disagreement and the same three parts survive.

1. **A physical twin.** A *specific* real thing. Not a class of things – not "a potted basil" but *this* pot, on *this* desk, with the probe pushed into *this* soil. Chapter 1 pinned the term and it is unchanged. The Non-Physical Systems section widens what "physical" may mean; it never widens "specific".
2. **A digital object (DO).** The software side: models, stored state, data and services that represent that one PT. 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 the two, in one direction or both, automatically or carried by a person.

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 item is where the whole section happens.

### The data-flow test

Here is the definition this book uses from now on. It comes from a categorical review of twins in manufacturing, which noticed that authors were writing *digital model*, *digital shadow* and *digital twin* interchangeably while meaning systems whose data moved in quite different ways, and gave each of the three its own name [7]. Trace the data. Ask about each direction separately, and ask whether the flow is *automated* or needs a person to carry it. In particular, look at the flow *back*: in two of the three classes it exists, and the only thing that separates them is who carries it.

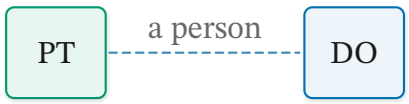
> **Digital model (DM).** A physical twin and a digital object both exist, and there is no automated data flow in either direction. All exchange is manual: a change of state in the PT has no direct effect on the DO, and a change in the DO has no direct effect on the PT.
>
> **Digital shadow (DS).** The flow from the PT to the DO is automated; the flow back is not. When the PT's state changes, the DO follows on its own – but

whatever travels from the DO back to the PT is carried by a person.

**Digital twin (DT).** The data flows are automated in *both* directions. The PT changes the DO, and the DO changes the PT.

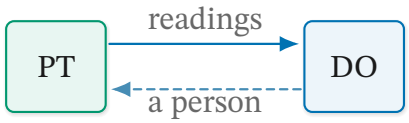
Pause on the shadow's return path, because it is the part most often drawn as missing. It is not missing. Somebody reads what the shadow shows and does something to the PT, or nobody would pay for the shadow; the flow from DO to PT is real, and a person is the channel. What the shadow lacks is the *automated* version of that flow, and that single word is the whole distance to a twin.

The split has been picked up widely enough that you will meet it under several phrasings, and they agree. One restatement puts it as manual exchange in both directions being a model, automatic physical-to-digital transfer with no reaction from the digital side being a shadow, and bilateral communication being a twin [8]. Another supplies the everyday instances this chapter will lean on: a computer-aided design (CAD) model of a car is a digital model, the mileage and speed on the car's dashboard are a digital shadow, and a twin needs the automated flow back [9]. An architecture-pattern catalogue frames the same three as design alternatives – data moved by manual intervention; sensors providing the automated flow inward; the two objects connected in both directions [10] – and a reproducibility study restates them as the standard subcategories by level of integration with the PT, noting in passing that the firms it studied mostly ran the first two [11]. That much agreement across independent groups is why this is the definition worth memorising. Figure 2 is the whole distinction on one page: three pictures that differ only in which arrows are automated.




(a) digital model

no automated flow either way; whatever moves, somebody carries




(b) digital shadow

automated flow inward; the return exists, and a person carries it




(c) digital twin

automated flow both ways; the DO knows and acts

Figure 2: 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. In (b) the return arrow exists and a person carries it; the automated version of that same arrow, in (c), is the subject of the rest of this section.

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, whose only way back to the PT runs through a person, is a shadow.

### 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. The return flow, if any, is
             carried by a person.
      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 two-parameter drying model fed automatically from a soil probe and wired to a relay is a digital twin. The test is indifferent to how impressive the software is, which is exactly why it is useful: sophistication is what vendors compete on, and it tells you nothing about what the system will do.

**"Without a person carrying it" is the load-bearing clause.** The Misconceptions section 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 deployment, not a product.** The same codebase can be a shadow in one installation and a twin in another, if the second enables an output path the first does not. Classify what is deployed.

### Why the line is drawn here

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 configuration. You cannot answer "is the model faithful enough?" that way; that takes a measurement campaign and a later chapter.

*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 the next subsection 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.

With that, Chapter 1's loose working sentence is retired and replaced:

> **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*. This one states what makes it a twin rather than a shadow. The book uses both, at different jobs: the earlier sentence when arguing value, this one when classifying a system.

### The return arrow is a control loop

Look again at panel (c) of Figure 2. Readings go up; commands come down; the commands change the PT; the next readings reflect the change. That is a feedback loop, and a twin is one by construction. The engineering-of-digital-twins literature names the difference between a twin and a shadow in exactly these terms: a twin can support reconfiguration of the PT with some degree of autonomy, which is what lets the PT respond to changes in its environment, and the open problem is how to *close the loop* well [12]. The same loop appears in the self-adaptive-systems reading of twins: the DT monitors the physical system, updates its internal model of it, and adjusts it through controllers to hold given requirements – the twin as the managing system, the PT as the managed one [13].

Two things follow from reading the return arrow as control rather than as "output".

The first is an ordering. You build enough confidence in the twin's picture to trust conclusions drawn on it, and only then let it inform decisions that change the real world – and only where that trust has been earned [5]. Shadow first, twin second, is not timidity; it is the order in which the evidence arrives.

The second is that the return arrow carries obligations the inward arrow never did. Figure 3 draws the loop as the rig will run it, and puts a box on the return path that panel (c) of the earlier figure left out: the **command path**, the checks a command passes before it reaches the PT. The US National Academies' report on twins makes the same point from the definition side: the bidirectional interaction may run in real time, as for an autonomous vehicle, or slowly, as for post-flight updating of an engine twin, and the decision at the end of it may be automatic or have a human in the loop – but it is always a decision that acts [14].

Table 1 is the ledger of what changes. Read it as the scope difference between two quotes for the same rig.

Table 1: What changes when the loop closes. Every row of the right-hand column is an obligation the left-hand column never had.

| | 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, caps, interlocks |
| Reversibility | Not applicable | Must be designed for; some actions cannot be undone |
| Staleness | Shows an old number | Acts on an old number; needs a bound |
| Concurrency | Not applicable | Another writer may be changing the same state |
| Testing | Against recorded data | Against the PT, or a trusted stand-in for it |
| Failure of the connection | Stale display; degrade gracefully | Must fail safe, which you have to define |
| Credibility evidence | Useful | Mandatory |

The row to read first is the top one, because every other row follows from it. Once the worst case is "does something false", accountability moves from a person to your software, and the command path, the staleness bound, the fail-safe behaviour and the credibility evidence all arrive at once. Four consequences for how you work:

- **The safety question becomes yours.** Before writing the command path, write down the worst thing the twin can command and what physically prevents it. Often the right answer is a limit in the controller's firmware that the twin cannot raise, rather than a check in the twin.
- **You need a state you can trust, not only data you can read.** A shadow can be sloppy about whether its state estimate is right, because a person reads the number and applies judgement. A twin cannot.
- **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 reading that triggered it. All of these are ordinary distributed-systems problems wearing physical consequences, which is good news for a reader with your background.
- **The incremental path is the normal path.** Model, then shadow, then twin is a sensible order, and each stage pays for the next. It is Chapter 1's lean argument in this chapter's vocabulary.

### Other definitions you will meet

You will be handed definitions that are not this chapter's. Three are common, and the professional move is to reconcile them with the data-flow test rather than argue, because they are mostly answering different questions.

**The aerospace one.** NASA's, from a 2010 roadmap, is quoted across the literature in its

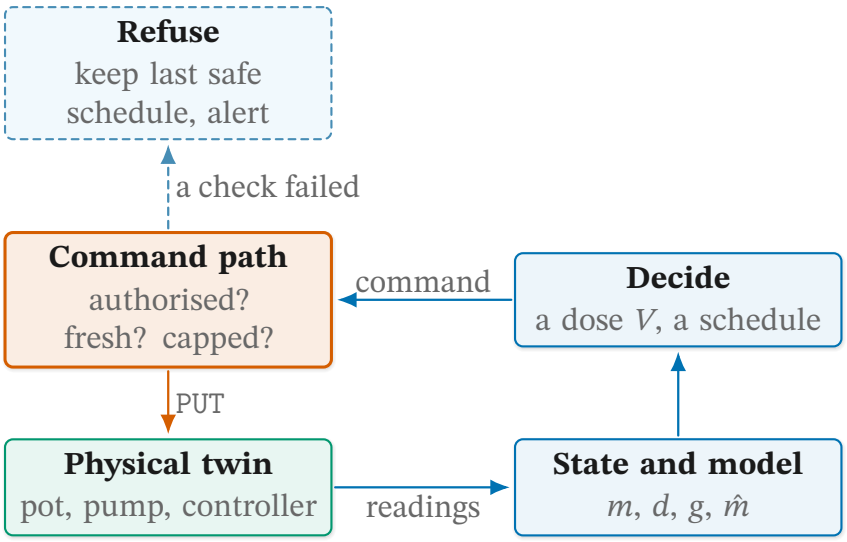



Figure 3: The twin as a control loop. Readings travel up the right-hand side, a decision comes back down the left, and everything in the command-path box is work a shadow never had to do. The dashed branch is what the twin does when a check fails.

own words: a twin is "an integrated multiphysics, multiscale, probabilistic simulation of an as-built vehicle or system that uses the best available physical models, sensor updates, fleet history, etc., to mirror the life of its corresponding flying twin", as reproduced in [2]. It is optimised for a domain where being wrong kills people and where the twin's job is life prediction on one airframe. Note "as-built" and "its corresponding flying twin": the referent is specific, which agrees with this chapter. Read it as a description of the high-fidelity end of a spectrum, not as the entry requirement; applied as an entry requirement, almost nothing in industry qualifies, including systems that are unambiguously twins by the data-flow test and are paying for themselves.

**ISO 23247's.** The manufacturing standard defines a twin as a "fit-for-purpose digital representation of an observable manufacturing element with a means to enable convergence between the element and its digital representation at an appropriate rate of synchronization" [15]. Three phrases are doing work. *Fit for purpose* is Chapter 1's fidelity principle in standards language, and the standards analysts say as much [4]. *Observable* rules out twinning what you cannot measure. *An appropriate rate of synchronisation* is the staleness row of the table above, promoted to the definition. The standard's "observable manufacturing element" is this chapter's PT under another name, and a recent survey lines the two vocabularies up side by side [9]. Take it as a refinement of the data-flow test that adds a rate question, not as a rival.

**The consolidated academic one.** A generalisation exercise across 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", and shows how many different words the literature uses for each of those four parts – "counterpart", "mirror", "clone" for the representation; "asset", "process state", "vehicle" for the physical system [16]. It is the most compatible of the three and the least operational: "exchange of information" is the data-flow test's axis stated without a direction, so it will not settle an argument about a dashboard.

Table 2 puts the three beside the test this book uses, on the one question a project manager

will actually ask.

Table 2: Four definitions on one question. The row that answers it is the row this book uses, and that is the only reason it was chosen over the others.

| Definition | Asks | Answers “is my dashboard a twin?” |
|---|---|---|
| Data-flow test (this book) | Which directions are automated? | Yes – and the answer is no |
| Aerospace | 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; the direction is unspecified |

The pattern to read off the table is in the last column: only the first row gives a yes-or-no answer that two engineers could check against a deployment diagram. The others are not wrong. They answer questions about fidelity, purpose and exchange that this book takes up later, one chapter at a time.

## Non-Physical Systems

Everything so far has had a pot, a pump or an airframe at the end of the arrows. The word “physical” in *physical twin* invites that reading, and for most of this book it is the right one. But the two-question test never mentions matter. It asks whether state reaches the digital object without a person carrying it, and whether the digital object’s output changes the referent without a person carrying it. Anything with a state that changes over time and can be observed and acted on can stand in the PT’s place. This section walks through three families of such referents that a software engineer will meet, and shows that the test survives each move unchanged.

The literature has already made the move. A vision paper on how twins evolve prefers the phrase **actual twin** for the object, system *or process* a twin represents, precisely so that “physical” stops doing work it was never meant to do [17]. An earlier survey says the real twin may be physical or conceptual, and lists humans, urban buildings and industrial or business processes among the referents alongside vehicles and manufactured products [18]. This chapter keeps writing PT, because the book has used it for two chapters and the rig is a real pot, and reads it broadly from here on. Figure 4 shows the shape the rest of this section fills in: the same loop three times, with a running service, a hospital ward and a city in the PT’s slot, and the class decided each time by who carries the return arrow.

### Software systems

A running software system has state – queue depths, error rates, deployed versions, the configuration in force – that changes minute by minute and is observed through telemetry. It

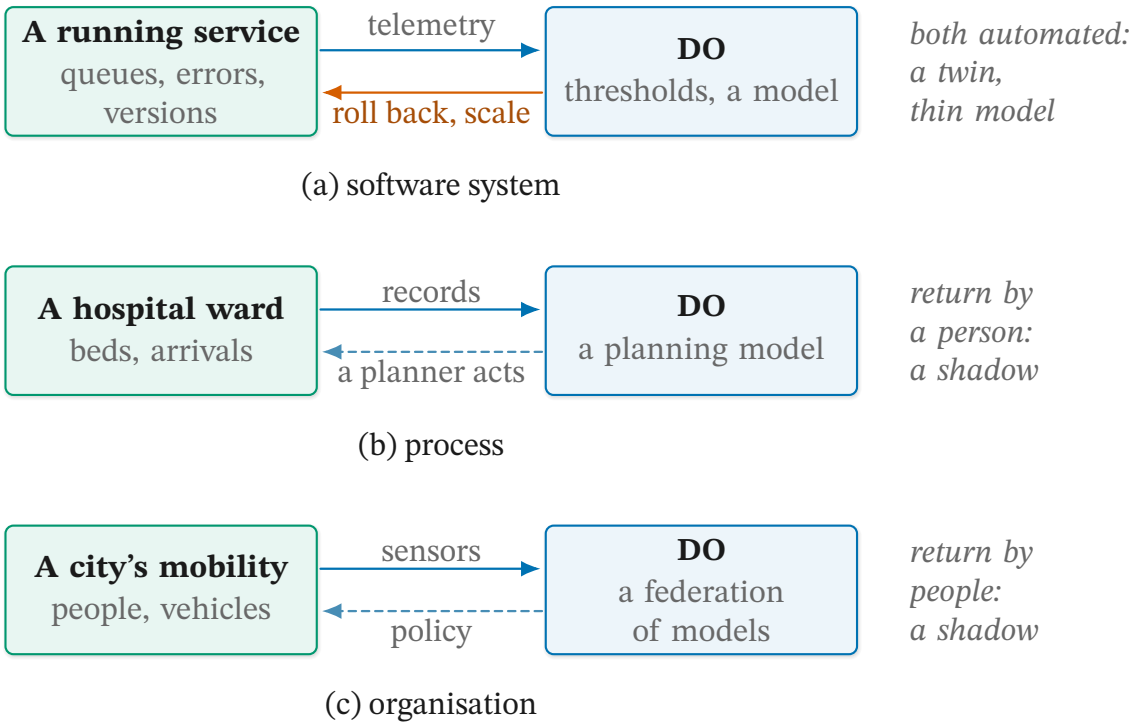



Figure 4: Three non-physical referents in the physical twin's slot. The inward arrow is automated in all three; only in (a) is the return arrow automated too, so only (a) is a twin.

also has levers: a deployment can be rolled back, a feature flag flipped, a replica count raised. So the two questions apply. A dashboard of a service's metrics is a digital shadow of that service, by exactly the argument the Misconceptions section makes for a car's dashboard. An autoscaler that reads the same metrics and changes the replica count is, structurally, a digital twin of the service – though a very thin one, because its "model" is a threshold rather than a prediction.

Two published lines of work make this concrete. The first turns the telescope around and applies twins to the *software engineering process*. A recent position paper starts from how much of a team's DevOps activity is now recorded as data, asks what a rich enough model of that record would be, answers that it would be a twin of the software engineering process itself, and rests the answer on the same data-flow model this chapter uses: the twin would *sense* data from the process, *reason* about it with internal models, and *act* on the process, for instance by suggesting code improvements [19]. Run the test yourself. Sensing from the pipeline without a person carrying it: yes. Acting on the process without a person carrying it: if the twin opens the pull request and a human merges it, that is a shadow with a human-in-the-loop actuation path, and the paper's own vocabulary lets you say so.

The second line runs the other way and treats the twin *as* software to be operated. One group states it as a definition – what a twin *is*, to them, is a piece of software: one that uses models and data about a cyber-physical system to represent, predict and prescribe its behaviour for a specific purpose, used at operations time as the counterpart of the design model used at development time [20]. Under the name TwinOps, others proposed unifying model-based engineering, twins and DevOps in one workflow for cyber-physical systems, on the grounds that DevOps already covers the path from development through integration to run-time monitoring and improvement [21]. The point for this chapter is the one that a

software engineer will find comfortable: whatever the referent, the digital object is a service, it is built, deployed, monitored and changed like one, and its classification says nothing about how it should be shipped.

### DevOps for a building's twin

The DevOps thread reaches physical referents too, and a building is the instructive case because a building is physical, large and slow, and its twin is neither. A framework for the engineering of twins of *built assets* – a multi-functional room, a climatic chamber – starts from the observation that a twin is complex software that must be continuously modified to meet evolving requirements, and supplies the continuous-integration and continuous-delivery infrastructure to do so [22]. A campus twin from another group brings a university's buildings, their day-to-day running and what-if simulations under one roof, and makes a point this chapter needs: the fixed description of a building has long lived in its information model, while the live feeds from its sensors and its automated controls have had nowhere to live and nobody acting on them [23]. Run the test on that campus twin: sensors deliver state automatically, so Q1 is yes; whether any setpoint is written back without a person decides Q2, and the paper describes a preliminary platform focused on energy use, which reads as a shadow. Two things to carry away. A building's twin is a software product with a release cadence measured in weeks, wrapped around a referent whose own changes are measured in decades. And DevOps applied to the twin's software is a different thing from a twin of a DevOps process: the same practice, pointed at different referents, and the test tells them apart.

### Processes, organisations, cities

The widest move is to a referent with no fixed hardware at all. A hospital's patient flow is a process: patients arrive, need beds, are discharged, and the ward's state is a set of occupied and free beds that changes hourly. A twin built at the University of Oslo turns that stream of arriving patients into a stream of optimisation problems, solved by a constraint solver over an executable formal model, to plan ward needs and explore what-if scenarios [24]. A related effort mirrors a hospital's operating suite – rooms, resources, staff – as an ecosystem of twins integrated with the hospital's legacy systems through a healthcare data standard [25]. Neither has a pump. Both have a referent with observable, changing state, both read that state automatically, and both hand their output to a planner, which by the test makes both shadows with a human-in-the-loop actuation path. That is not a criticism; it is the correct class for a system whose actuator is a hospital administrator.

The same holds one level up. A supply chain is a network of organisations, and twins of supply chains let managers build alternate realities of their network and explore its vulnerabilities, a digital form of wargaming for resilience [26]. A city's mobility is travellers, their vehicles and the roads and rails under them, all at once, and one federation of twins built to weigh up interventions in it was shaped by a plain fact: automating such decisions conflicts with how planning is actually done – so it keeps human-in-the-loop control throughout and writes down which steps are automated and which are human decisions [27]. That last design choice is the Misconceptions section's argument arriving from the other direction: when the referent is a society, the person in the loop is a requirement, not a gap.

### What the move costs

Widening the referent costs nothing at the level of the test and a great deal at the level of the model. A pot's state is one number from one probe. A hospital ward's state is hundreds of records in a legacy system, some of them wrong. A city's is contested. The National Academies' definition of a twin explicitly covers "the engineered world, natural phenomena, biological entities, and social systems", and in the same breath insists on *predictive capability*: a twin "must be able to issue predictions beyond the available data to drive decisions" [14]. The less the referent resembles a machine, the harder such a prediction becomes to make, and to believe, and the more often a working system's honest class turns out to be *shadow*. Keep the two facts separate. The test does not care what the referent is made of. The model does.

## Types of Digital Twins

Once the referent can be anything with observable state, it helps to have names for the kinds of referent people actually twin. The literature has several such schemes. One review catalogues two: a shop-floor scheme of *product*, *process* and *operation* twins, and a more general one of *object*, *process* and *phenomena* twins [28]. Another review's definition admits three kinds of referent at once: something made, something offered or something done – product, service, process [29]. This section uses six labels that cover what a software engineer will meet – product, process, lifecycle, biological, ecological, sociological – and makes one point about all of them before it starts.

**Every one of these is orthogonal to the data-flow test.** A type names *what* is twinned. The test names *how data moves*. A product twin can be a model, a shadow or a twin; so can a process twin; so can a twin of a lagoon. The professional sentence states both: "a process twin of the ward, currently operating as a shadow." That sentence is precise, and it is the kind of sentence that ends a circular meeting. Figure 5 draws the relationship: six kinds of referent on the spokes, one test at the hub.

**Product twins.** The referent is one manufactured thing: this pump, this turbine, this airframe. The rig's plant twin is a product twin in this sense – the referent is one pot – and so is every twin the aerospace definition had in mind. The observable state is the thing's condition; the return arrow, if there is one, changes how the thing is operated or maintained. Most of Chapter 1's deployment families were product twins.

**Process twins.** The referent is an activity with a beginning and an end that repeats: a production line's cycle, a fermentation, a surgery's turnaround, a patient's stay. The standards analysts note that a twin can be created for processes as well as products, with simulations run on real-time process data [4], and the hospital twins of the previous section are process twins. Their state is a schedule or a queue rather than a temperature, and their actuator is usually a person, which is why they classify as shadows so often.

**Lifecycle twins.** The referent is one thing followed *across* the phases of its life rather than in one of them. A review of twins along the product lifecycle splits that life into design, production, use and disposal, and finds that most published twins live in the middle two phases, as prototypes focused on real-time observability, with the design and disposal phases still thin [30]. Two complications make lifecycle twins harder than they sound. The referent itself has phases – a machine in commissioning sends telemetry at a different rate and range

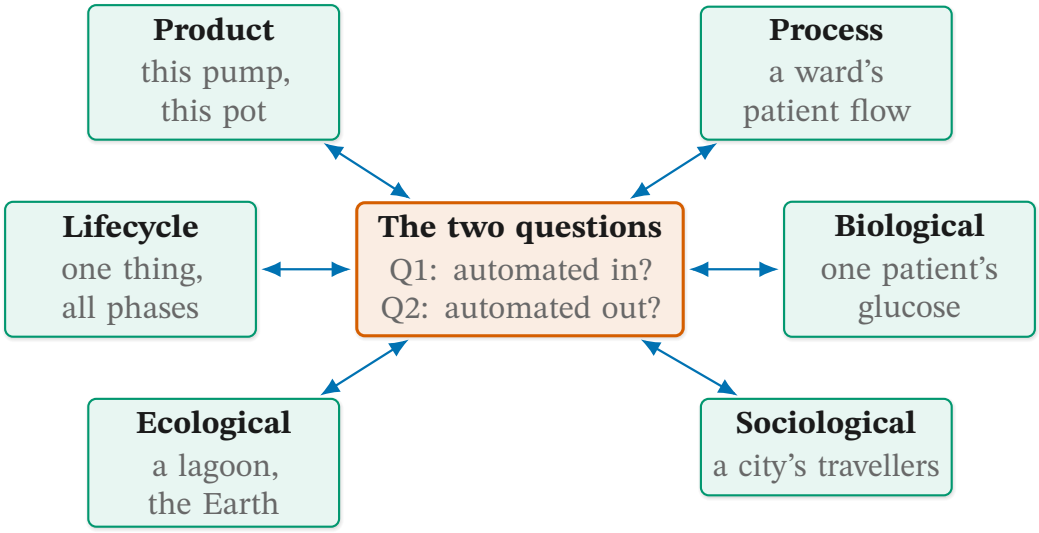



Figure 5: Six kinds of referent, one test. Whatever sits on a spoke, the two questions at the hub decide whether it is a model, a shadow or a twin.

than one in service, and a twin that does not model those phases may read an expected change as a failure [31]. And the twin has a life of its own that must be kept in step with the referent's; the engineering literature treats that continuous, mutual evolution as the core problem of building twins at all [17]. The same idea reaches software: a twin could accompany a software system through its own life cycle, and with a product line there could be one twin for the reference design and one for each product built from it [32].

**Biological twins.** The referent is an organism or part of one, most often a patient. The claims here run ahead of the deployments, and a careful reader should hold the two apart. On the deployment side, a roadmap for immune-system twins points to twins of the human heart in diagnosis and to the artificial pancreas, which manages insulin for Type I diabetic patients, as operational medical twins [33]. Run the test on the artificial pancreas: a glucose sensor delivers state automatically, a model decides a dose, a pump delivers it. Both arrows automated, one specific patient. It is a digital twin by this chapter's definition, and it is one of the few in medicine that is. On the aspiration side, a European position paper sets out a *Virtual Human Twin* infrastructure for developing and validating healthcare twins and for personal health forecasting [34], and a perspective on personalised medicine argues that data-driven methods alone cannot deliver it and must be complemented by model-driven patient twins [35]. The data caveat is severe: twins that track a person's health over time need longitudinal data, which is scarce, fragmented across institutions and noisy [36]. Most of what is called a patient twin today is, by the test, a digital model or a shadow, and the perspective papers say as much between the lines.

**Ecological twins.** The referent is an ecosystem or the planet. A marine twin of a Mediterranean coastal lagoon models the lagoon's ecosystem, uses learned models to emulate its hydrology, and runs hypothetical scenarios for managers, motivated by human and climate pressures on the coast [37]. At the largest scale, the European Commission's Destination Earth programme is building a high-precision twin of the Earth system for climate adaptation and disaster-risk management, one of several scientific twin initiatives that have

followed the concept out of industry into Earth-system science and biomedicine [38]. A collection of commentaries on Earth twins adds the sobering condition: to be more than big-data atlases, such twins need a new Earth-system simulation and observation capability, and they must model the Earth *and* the humans acting on it together [39]. Apply the test and the answer is instructive. Observations reach the digital object automatically, so Q1 is yes. Nothing the digital object outputs changes the ocean without a person, so Q2 is no. A twin of Earth, in this chapter's vocabulary, is a very large digital shadow whose only actuator is policy – and its designers know it, which is why the human-interaction problem is the one they name.

**Sociological twins.** The referent is people in the aggregate: a city's travellers, a supply network's firms, a hospital's staff. The urban-mobility federation of the previous section is the clean example, and its defining design decision – human-in-the-loop control throughout, with the automated and human steps written down [27] – is the right one for a referent that can read the twin's output and change its behaviour in response. The mobility twin from another group goes one step further and gives the human a twin of their own, a *Human Digital Twin* that classifies driver type and feeds a personalised cruise controller, alongside twins of the vehicle and the traffic [40]. When the human is the referent, the return arrow is a recommendation, and a recommendation a person may ignore is, by the test, a person carrying the flow.

Table 3 collects the six, with the one question to ask of each.

Table 3: Six kinds of referent. The last column is what the test usually returns for published systems of that type; none of the types forces a class.

| Type | Referent | What the inward arrow carries | Usual actuator | Class most often met |
|---|---|---|---|---|
| Product | one manufactured thing | condition, load, position | a controller or a maintainer | shadow or twin |
| Process | one repeating activity | schedule, queue, cycle time | a planner | shadow, human in the loop |
| Lifecycle | one thing across its phases | phase, configuration, history | varies by phase | model early, shadow later |
| Biological | one organism or organ | physiological signals | a clinician; rarely a device | model or shadow; the artificial pancreas is a twin |
| Ecological | an ecosystem, the planet | observations, satellite data | policy | shadow |
| Sociological | people in the aggregate | movements, transactions | the people themselves | shadow, human in the loop |

Read down the last column and one pattern stands out: the further the referent is from a machine with an actuator, the more often the honest class is *shadow*. That is not a failure of the field. It is the test doing its job, and telling you where a person still has to stand.

## Misconceptions on Digital Twins

Negative definitions teach faster than positive ones, because the reader arrives with the wrong ones already installed. Five things a digital twin is not, each with the tell that exposes it.

### 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 a rendered plant floor photographs better than a residual plot. Visualisation is genuinely valuable: a good picture shows a person a shape in the data that a table of numbers hides – a drift, a repeating pattern, two channels moving together – and with it the trouble that shape may be announcing; the useful forms run from plain charts through animated schematics to three-dimensional scenes [41]. Note the verb: it *presents*. Presentation is a service the twin offers, and a later chapter 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 – which the taxonomy's own examples file under *digital model* when the feed is manual and *shadow* when it is not [9].

### Not a monitoring dashboard

Chapter 1 said, of the *monitor* pattern, that many systems sold as twins stop there. Settling whether that counts:

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

Run the test. Q1: does the PT's state reach the digital object automatically? Yes; that is what a dashboard is. Q2: does the digital object's output change what the PT does, without a person carrying it? No. A dashboard's output goes to a human retina. Digital shadow. The taxonomy's own illustrative shadow is exactly this – the mileage and speed on a modern car's dashboard [9].

Three things follow, and the third is the one that matters commercially. *This is not an insult.* Chapter 1's worked example, the dosing-rig fault detector with a nine-month payback, is a shadow. Shadows earn money, and reviews of the field consistently find more published work on models and shadows than on twins proper [7]. One review went further and measured it: over half of the papers it examined described a model or a shadow while claiming a twin, so it distinguishes the *claimed* twin, the *described* twin and the *actual* twin, and finds that most claimed twins are, in their case studies, models or shadows [28]. *But you must not sell a shadow as the other thing.* The category difference is real, the customer's expectation attaches to the word, and the gap is exactly the right-hand column of the obligations table. *And the honest sentence is short.* "This is a digital shadow. It will tell

you within a check interval that a dose did not land. It will not reschedule the dose. Making it do so is a separate increment, and here is what that costs."

### Not 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 – which is not a loophole but the definition: a shadow's return path is the one a person carries.

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, the human applies judgement the digital object does not have and rejects commands that are plainly wrong; the system's response time is bounded by human availability – shifts, weekends, holidays – and that bound is usually the thing being bought; and accountability rests with an identified person. Remove the human, and all three move into your software. Those three differences are not vocabulary. They are the difference between two systems with different failure modes, different response times and different liability, and the test tracks that difference under a word that sounds pedantic.

Be precise rather than dogmatic. A survey of twins used for AI simulation sorts what it found into three styles – *twin*, *shadow* and *human in the loop* – and reports that the autonomous style dominates, with a minority where a person supervises or acts [42]; a mobility twin has been demonstrated with the driver deliberately kept in the loop and given a twin of their own [40]; and one of the taxonomy's own restatements observes that unless a twin is fully autonomous, humans will need to interact with it [8]. The professional formulation is therefore not "that is only a shadow" but **"a digital shadow with a human-in-the-loop actuation path"** – which says exactly what exists, credits what it delivers, and makes the remaining gap to a twin visible.

### Not a simulator

The first section 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 on the desk, in the state it is in now. A simulator with no referent and no live data is a digital model in this chapter's sense, however good it is. The engineering literature's warning bears repeating – a model plus a run of it is still not a twin [3] – and the standards analysts agree on where the line falls: in the live measurements and the path back to the sensors, not in the model [4].

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. This is not a criticism of the simulator, which is usually the most valuable asset the project has and the thing a twin gets *built around*.

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

Storage is not a twin: a historian holding five years of 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 a later chapter covers what changes when a learned model sits there instead of the pot's two equations.

The general form of all five: 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.

## Classification of the Plant Demonstrator

Chapter 2 put the rig on the desk and wrote down its model. This section classifies it – at four stages of its life, with the reasoning at each step, using nothing but the two-question test and the rig's own API. It is the chapter's fully worked example; the Faded Examples section then takes the scaffolding away.

Two facts from the previous chapter carry the argument. The PT is the pot, plant, soil, pump, tubing and the Raspberry Pi controller. The controller exposes a REST API on port 8099 whose endpoints include:

```
GET /sensing/{unit}/{parameter}   readings; limit, since_timestamp
GET /actuation/{unit}/watering_events  doses delivered
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-pump setting is a recognised exemplar for exactly this kind of reasoning. A published greenhouse twin has, on its physical side, potted plants with probes in the soil and pumps to water them, and on its digital side a loop that both steers the pumps and keeps the twin's own models honest [43]; the same group's later work on the same rig reads the twin and its physical counterpart together as a self-adaptive system in which the twin monitors, updates its model, and adjusts the physical system through controllers [13]. The rig in this book is a smaller member of that family. Nothing below is a toy argument about a toy.

### Stage 0: today, as shipped

**What exists.** The controller samples its sensors, writes readings to the time-series database on the Raspberry 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 one. Recall the three ingredients: the PT is *the pot, plant, soil, pump, tubing and the controller*. The controller's own logging is the PT 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.

### Stage 1: a model in a notebook

**What exists.** You export a few weeks of readings, fit the two parameters $d$ and $g$ of the pot's model in a notebook, and produce a chart of predicted against measured moisture. You do this by hand, once. The simulation worked in the first section – $m(0) = 44$ at 06:00, one dose, a predicted $m(12) = 42$ at 18:00 – is exactly what this notebook draws, against the reading the probe actually gave.

**Run the test.** Q1: does the PT'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 represent this pot at all, which is the cheapest thing to find out early and the most expensive to discover at Stage 2. Note that this stage is neither free nor skippable, and that its output is knowledge rather than software.

### Stage 2: the fault detector

**What exists.** A service that every $T_c$ hours calls `GET /sensing/plant_1/moisture` and `GET /actuation/plant_1/watering_events`, steps the model forward, computes the residual $r = \hat{m} - m_{\text{obs}}$, and raises an alert when a scheduled dose produces no rise in the reading. This is the previous chapter's *diagnose* increment: the reading that arrives after a dose is compared with the reading the model expected, and a residual that stays positive says less water is arriving than was asked for.

**Run the test.** Q1: state reaches the digital object automatically? **Yes.** The polling loop, over HTTP, with no person in it. Q2: does the digital object's output change what the PT does, automatically? **No.** The output is an alert to a technician, who reads it and, if the pot is dry, edits the schedule by hand. The return flow exists; a person carries it.

**Verdict: digital shadow.**

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

> **The payback-positive first increment is not a digital twin.**

That is not an embarrassment and not a bait-and-switch. It is the normal, correct shape of a first increment, and it is why the Misconceptions section 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 at the kickoff rather than at the acceptance review is the practical payoff of this whole chapter.

The technician's edit is the loop closing through a person, and it is what makes Stage 2 a shadow rather than a twin. The Misconceptions section gives you the precise phrase for it, *a digital shadow with a human-in-the-loop actuation path*.

### Stage 3: closing the loop

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

```
PUT /actuation/plant_1/update_schedule
```

with a schedule it computed itself – the target-moisture rule of the previous chapter, $V = \min\big(V_{\max},\ k\,(m_{\mathrm{t}} - m)\big)$, wrapped as a one-event schedule.

**Run the test.** Q1: yes, unchanged. Q2: **yes.** The digital object's output now reaches the PT 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 the obligations table 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 but the water tank's volume. The cap $V_{\max}$ in the twin helps; a hard cap on total daily dose belongs in the *controller*, because the twin is the component whose correctness is in question.
- **Authorisation.** The endpoint accepts 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.** The `since_timestamp` parameter 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, and the bound follows from the decision – doses are hours apart, so a bound well inside one check interval – not from the phrase "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 the later, architectural part of this book is the deep part for a reader with your background. The command-path box of Figure 3 is where all five live.

### The classification table

Table 4 collects the four stages, and Figure 6 draws them, so the shape of the climb is visible: each stage adds exactly one arrow, and the last stage adds none – it automates the one a person was carrying, and that alone changes the class to *twin*.

Table 4: The rig at four stages. Between stages 2 and 3 neither the model nor the existence of a return flow changes; only who carries that flow does.

| Stage | What exists | Q1: automated in? | Q2: automated out? | Class |
|---|---|---|---|---|
| 0 | Controller only, fixed schedule | no digital object | – | none |

| Stage | What exists | Q1: automated in? | Q2: automated out? | Class |
|---|---|---|---|---|
| 1 | Hand-fitted model in a notebook | no | no | Digital model |
| 2 | Polling fault detector; a technician acts on its alert | yes | via a person | Digital shadow |
| 3 | Detector writes the schedule with `PUT` | yes | yes | Digital twin |

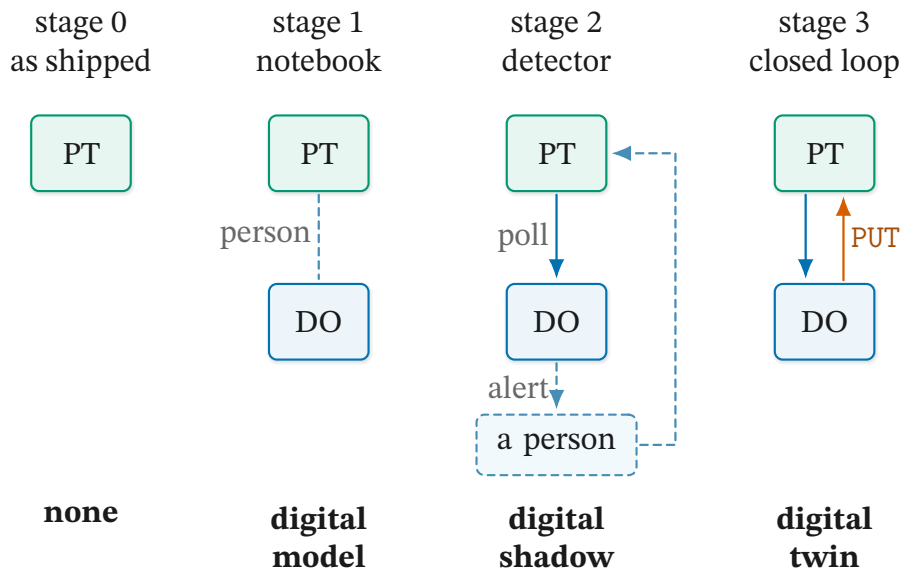



Figure 6: The rig's climb. One arrow per stage; in stage 2 the return arrow already exists and a person carries it. The jump that matters is from stage 2 to stage 3, where that arrow becomes automated, and it is a jump in obligation, not in modelling effort.

Read the table's second column against its last. The model at Stage 3 is the same two-parameter drying model as at Stage 1. What changed is who is allowed to act on it, which is why the obligations table is the expensive part of the picture rather than the box marked DO.

One last observation, and it is the rig's own doing. Its documentation names two kinds of user, and describes the second, the *remote connecter*, as either a person or a computer running a digital twin. The PT was designed with the Stage 3 seam already cut. Recognising that seam in somebody else's system, before you write any code, is the skill this chapter has been teaching.

## Faded Examples

Same method, less scaffolding. For each system, name the PT, the digital object, answer Q1 and Q2, and classify. The first is worked in full, the second is started and left for you to finish, and the third is posed as a problem.

**Example 1, worked in full: 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.

*PT:* 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. Stop. **Digital model.** Note that the model is sophisticated and building-specific, and that changes nothing: it is the first property of the two-question test, flows and not sophistication. Note also what would change the answer. Wire the meters to the model and Q1 flips to yes: a shadow, with no change to the model at all. Let the model write setpoints to the building's heating controller and Q2 flips too, and the obligations table arrives with it.

**Example 2, one step left to you: the offshore turbine anomaly service**

Vibration and temperature stream from an offshore wind 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.

*PT:* turbine number 12. *Digital object:* the model, 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 the service 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 the service 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):* the Misconceptions section supplies a phrase that describes what exists without conceding the label.

**Example 3, posed as a problem: 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.

Classify it, unaided. Then answer two harder questions. Does the weekly retraining change the classification? And if the operators exercise their override twice a year, does the system stop being a twin on those two days?

## Summary

Back to the objectives this chapter opened with.

1. **Model, simulation, simulator are three things.** A description; one execution of it from a start state; the software that executes it. The rig's two equations are a model,

the run from $m(0) = 44$ to $m(12) = 42$ is a simulation, and the dozen lines that step it are a simulator. The sentence that settles most arguments: a twin needs a referent, a simulation does not.

2. **Classification is a question about data flows.** No automated flow in either direction: digital model. Automated physical-to-digital, with the return carried by a person: digital shadow. Automated in both directions: digital twin [7]. Two questions, answerable from a deployment diagram, indifferent to how sophisticated the software is – and that indifference is the feature.
3. **The hard cases have answers.** A dashboard is a shadow, and that is not an insult. A human closing the loop leaves it a shadow, best described as *a digital shadow with a human-in-the-loop actuation path*. A simulator with no referent is a model. A referent need not be a machine – a software process, a hospital ward, a city, a lagoon – and the test is unchanged; only the model gets harder. Twins sort by referent into product, process, lifecycle, biological, ecological and sociological types, and every type can be a model, a shadow or a twin.
4. **Crossing the boundary changes the project, not the feature list.** The return arrow is a control loop. A wrong shadow displays something false; a wrong twin does something false. Authorisation, a staleness bound, concurrency, reversibility, fail-safe behaviour and credibility evidence all arrive at once, in the command path, and trust is earned on the shadow before the twin is allowed to act.
5. **The rig classifies cleanly, and the result is uncomfortable in a useful way.** Stage 0: no digital object. Stage 1: digital model. Stage 2, the payback-positive fault detector: digital *shadow*. Stage 3: one `PUT /actuation/{unit}/update_schedule` makes it a digital twin, and that single call drags five distributed-systems obligations in behind it.

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

## Exercises

Solutions or hints follow the exercises; each says which. The objective each exercise serves is given in brackets.

**Exercise 1 (Objective 2, 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.

**Exercise 2 (Objective 1, easy).** A colleague says "we already have the model of the pot, so the twin is mostly done." Name three distinct things that sentence might mean, and the one question you would ask to find out which.

**Exercise 3 (Objective 2, easy).** The six value patterns are monitor, diagnose, predict, decide, control and certify. 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.

**Exercise 4 (Objective 3, medium).** Complete Example 2 of the Faded Examples section, parts (a) to (c).

**Exercise 5 (Objective 3, medium).** Do Example 3 of the Faded Examples section, includ-

ing both harder questions.

**Exercise 6 (Objective 3, medium).** A continuous-integration server runs a team's test suite on every commit, and a bot posts a comment on the pull request when a test that passed on the main branch fails. Classify it. Then: the same bot is given permission to revert the commit automatically. What class is it now, and which three rows of the obligations table have now become the team's problem?

**Exercise 7 (Objective 4, medium).** For the rig at Stage 3, write the staleness rule: the maximum age of the moisture reading at which the twin is still allowed to write a schedule, and the reasoning that justifies it from the check interval $T_c$. State what the twin should do when the reading is older than that.

**Exercise 8 (Objective 4, medium).** The Stage 3 discussion 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.

**Exercise 9 (Objective 3, medium).** Pick one of the six types in the Types of Digital Twins section and construct, in three sentences each, a system of that type that is a digital model, one that is a digital shadow, and one that is a digital twin. The three should share a referent.

**Exercise 10 (Objectives 2 and 4, hard).** Take a system you have personally built or operated that talks to hardware, or that acts on a running software system. Classify it with the two-question test. Then: if it is a shadow, what exactly would closing the loop oblige you to add? If it is a twin, which rows of the obligations table did the project actually discharge, and which did it skip?

**Exercise 11 (Objective 3, hard).** The Digital Model to Digital Twin section claims the three outside definitions "mostly answer different questions". Argue the opposite: construct a concrete system that one of the three would call a digital twin and the data-flow test would not, and say which definition you would use in a contract and why.

**Exercise 12 (Objectives 2 and 4, hard, open-ended).** Your company is about to sign a statement of work whose 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 when the customer asks why you replaced their sentence with three of yours.

### Solutions and hints

**Exercise 1 (solution).** (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 that it is also the most technically demanding item on the list, which is the first property of the test again. (c) Digital twin, formally: sensor in, boiler command out, both automatic, one specific room. If that feels wrong, the discomfort is informative. The test is about structure, not scale, and a thermostat has the structure. What it is missing is any model beyond a setpoint, and how good the model is belongs to the *fidelity* question, not to classification.

(d) Neither: there is no specific PT. The planet is not a serial number, and a forecast is a simulation. A twin of one specific building, or of one specific lagoon, is a different matter.

**Exercise 2 (hint).** The first section. The three meanings are roughly: the two equations on paper (a description); code that steps them forward on demand (an executable model, with a simulator); and a *calibrated* executable model whose $d$ and $g$ have been fitted to this specific pot. The question to ask: "can I call it with today's readings and get an answer back for *this* plant?" The workflow those three collapse – conceptual model, executable model, verification, calibration – is the one the modelling-and-simulation literature lays out [2].

**Exercise 3 (solution).** Monitor: shadow. Diagnose: shadow. Predict: shadow; a forecast is still output to a person. Decide: shadow if it advises, twin if it acts, so the answer depends on the deployment, not the pattern. Control: twin, by definition of the pattern. Certify: often neither, since the PT may not exist yet, and a simulator with no referent is a digital model. The first pattern that *requires* a twin is *control*. Worth noticing: four of the six patterns are deliverable by a shadow, and the fifth sometimes is.

**Exercise 4 (solution, then hints).** (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 the 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.

**Exercise 5 (partial solution). 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 test is about data flows between physical and digital, not about how the model was obtained; it does change the credibility 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 the obligations table says a twin needs; exercising it is the system working as designed, not the system changing category.

**Exercise 6 (solution).** With the comment only: a digital shadow of the codebase-under-test – state (test results) reaches the bot automatically, the bot's output reaches a person. With automatic revert: a digital twin, structurally; the bot's output changes the referent without a person. The three rows: *worst case* (a flaky test now reverts a good commit), *concurrency* (the bot and a developer pushing to the same branch), and *reversibility* (a revert of a revert is not a no-op once other commits have landed on top). Note that this is Exercise 1(c)'s thermostat in software: the structure of a twin, a model no richer than a threshold.

**Exercise 7 (hint).** Doses are hours apart and the quantity the twin acts on changes on a scale of hours, so a defensible bound sits well inside one check interval – an hour or two when $T_c = 6$ – and is justified 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, and it is the dashed branch of the control-loop figure.

**Exercise 8 (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 between the read and the write. In the API: add a version or an entity tag to the schedule and have the controller reject a `PUT` that carries 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. Put the invariant where the state is.

**Exercise 9 (hint).** The easiest type is *product*. The same pump: its datasheet curve in a spreadsheet (model); a vibration feed into an anomaly service that emails a technician (shadow); the same service writing a speed limit to the pump's drive (twin). Check that your three share one referent and differ only in the arrows.

**Exercise 10 (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 rows of the obligations table specifically rather than gesturing at "more testing".

**Exercise 11 (hint).** The easiest construction uses the consolidated academic definition [16], 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: the test says nothing about whether the twin is any good, which is why the contract also needs the credibility clauses a later chapter supplies.

**Exercise 12 (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 says, in so many words, whether anything is written back to the line. That clause is the whole chapter.

## Further Reading

- *The taxonomy itself, first-hand:* [7] 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, [8], [9], [10] and [11]; for how often a paper's "twin" turns out to be a shadow, [28].
- *On there being no settled definition:* [6] for the twin/shadow boundary in manufacturing and [1] for the practitioner survey and the argument that a standard should fix it.
- *Model, simulation and simulator:* [2] traces the concept from classical modelling and simulation to twins and carries the workflow (conceptual model, executable model, verification, calibration) and NASA's definition; [3] and [5] open the book-length treatment this chapter's framing is closest to; [4] is the standards analysts' answer to "how is a twin different from simulation".
- *Definitions from outside the taxonomy:* [15] for ISO 23247's wording, [16] for the consolidation exercise and its lexicon, and [14] for the US National Academies' definition, which widens the referent and insists on prediction.
- *Closing the loop:* [12] for why the return arrow, and the autonomy it grants, is what separates a twin from a shadow, and [13] for the same loop read as a self-adaptive system.
- *Non-physical referents:* [17] for the "actual twin" phrase and [18] for conceptual referents; [19] for a twin of a software engineering process; [20] and [21] for the twin as software and TwinOps; [22] for DevOps on a building twin and [23] for a

campus twin; [24] and [25] for hospital processes; [26] for supply chains; [27] for urban mobility with the human kept in the loop.

- *Types, by referent:* [29] and [28] for the published type schemes; [30], [31] and [32] on lifecycle twins; [33], [34], [35] and [36] on biological twins, read together so that the deployments and the aspirations stay distinct; [37], [38] and [39] on ecological and Earth-system twins; [40] for a twin of the human in a mobility system.
- *The hard cases:* [41] on what visualisation is for, and [42] for human-in-the-loop as a named twin style.
- *The rig's relatives:* [43] for the greenhouse exemplar whose twin manages its pumps through a feedback loop; it is the closest published relative of the four stages worked above.
- *In the rig itself:* open the controller's REST API documentation at `/docs` and find the four endpoints listed in the Classification section. Then find the one `PUT` among them. That single route is the boundary this chapter is about, and everything the rest of the book builds sits on one side of it or the other.

## References


[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] T. Bergs, S. Gierlings, T. Auerbach, A. Klink, D. Schraknepper, and T. Augspurger, "The concept of digital twin and digital shadow in manufacturing," *Procedia CIRP*, vol. 101, pp. 81–84, 2021, Accessed: Jan. 21, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2212827121006612

[7] 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

[8] 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.

[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] 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.

[11] 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.

[12] 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.

[13] 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.

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

[15] 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.

[16] 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.

[17] T. Alskaif *et al.*, "Evolution at the Core of Digital Twin Engineering," Grand Rapids, USA: IEEE, Jul. 2025. doi: 10.5283/epub.77656.

[18] L. F. Rivera, M. Jiménez, N. M. Villegas, G. Tamura, and H. A. Müller, "The forging of autonomic and cooperating digital twins," *IEEE Internet Computing*, vol. 26, no. 5, pp. 41–49, 2021, Accessed: Jan. 12, 2025. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9325563/

[19] R. Kimmel, J. Michael, A. Wortmann, and J. Zhang, "Digital Twins for Software Engineering Processes," in *2025 IEEE/ACM 47th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, Apr. 2025, pp. 16–20. doi: 10.1109/ICSE-NIER66352.2025.00009.

[20] 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.

[21] J. Hugues, A. Hristosov, J. J. Hudak, and J. Yankel, "TwinOps - DevOps meets model-based engineering and digital twins for the engineering of CPS," in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, in MODELS '20. New York, NY, USA: Association for Computing Machinery, Oct. 2020, pp. 1–5. doi: 10.1145/3417990.3421446.

[22] 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.

[23] C. Lin, T.-Y. Dai, A. D. Dilsiz, D. Crawley, D. Niyogi, and Z. Nagy, "UTwin: A digital twin of the UT Austin campus," in *Proceedings of the 10th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*, Istanbul Turkey: ACM, Nov. 2023, pp. 282–283. doi: 10.1145/3600100.3626261.

[24] 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.

[25] 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.

[26] R. Bhandal, "Conceptualising the Application of Digital Twins in Supply Chain Management: A Path Towards Supply Chain Resilience," in *Digital Twin: Fundamentals and Applications*, S. Sabri, K. Alexandridis, and N. Lee, Eds., Cham: Springer Nature Switzerland, 2024, pp. 173–189. doi: 10.1007/978-3-031-67778-6_8.

[27] J. Li, J. Grübel, A. Nadi, M. Snelder, B. van Arem, and J. Gao, "Digital Twin Federation for Urban Mobility Assessment: A Functional Architecture for Low-Car Transformations in the Netherlands," 2025, Accessed: Oct. 25, 2025. [Online]. Available: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5370630

[28] 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.

[29] M.-E. Iliuţă, M.-A. Moisescu, E. Pop, A.-D. Ionita, S.-I. Caramihai, and T.-C. Mitulescu, "Digital Twin—A Review of the Evolution from Concept to Technology and Its Analytical Perspectives on Applications in Various Fields," *Applied Sciences*, vol. 14, no. 13, p. 5454, Jan. 2024, doi: 10.3390/app14135454.

[30] "Digital Twins along the product lifecycle: A systematic literature ..." *Digital Twin \textbar Open Access Publishing Platform*. Mar. 2023. doi: 10.12688/digitaltwin.17807.2.

[31] M. Picone *et al.*, "Harmonizing Physical and Digital Twins Lifecycles," in *2025 IEEE 22nd International Conference on Software Architecture Companion (ICSA-C)*, Mar. 2025, pp. 197–204. doi: 10.1109/ICSA-C65153.2025.00039.

[32] 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.

[33] R. Laubenbacher *et al.*, "Building digital twins of the human immune system: Toward a roadmap," *npj Digital Medicine*, vol. 5, no. 1, pp. 1–5, May 2022, doi: 10.1038/s41746-022-00610-z.

[34] M. Viceconti, M. De Vos, S. Mellone, and L. Geris, "Position Paper From the Digital Twins in Healthcare to the Virtual Human Twin: A Moon-Shot Project for Digital Health Research," *IEEE Journal of Biomedical and Health Informatics*, vol. 28, no. 1, pp. 491–501, Jan. 2024, doi: 10.1109/JBHI.2023.3323688.

[35] F. Emmert-Streib, S. Parkkila, R. Laubenbacher, A. Mannermaa, L. Hood, and O. Yli-Harja, "The role of digital twins in P4 medicine: A paradigm for modern healthcare," *npj Digital Medicine*, vol. 8, no. 1, p. 735, Dec. 2025, doi: 10.1038/s41746-025-02115-x.

[36] E. Katsoulakis *et al.*, "Digital twins for health: A scoping review," *npj Digital Medicine*, vol. 7, no. 1, pp. 1–11, Mar. 2024, doi: 10.1038/s41746-024-01073-0.

[37] Y. Ye, A. González-Vidal, A. Cisterna-García, A. Pérez-Ruzafa, M. A. Z. Izquierdo, and A. F. Skarmeta, "Advancing Towards a Marine Digital Twin Platform: Modeling the Mar Menor Coastal Lagoon Ecosystem in the South Western Mediterranean." arXiv, Sep. 2024. doi: 10.48550/arXiv.2409.10134.

[38] "interTwin: Advancing Scientific Digital Twins through AI, Federated Computing and Data."

[39] "The rise of digital twins," *Nature*. Mar. 2024. Accessed: May 27, 2024. [Online]. Available: https://www.nature.com/collections/feicjiideh

[40] 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/

[41] 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.

[42] 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.

[43] 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.