Chapter 7 – Should You Trust the Twin? Calibration, Credibility, and V&V

7.0 Before you start

Where we are. This is the fourth chapter of Part II, and it is the chapter every previous chapter has been writing cheques against. Chapter 1 said credibility is what the top value rung costs. Chapter 2 said credibility evidence is optional for a shadow and mandatory for a twin. Chapter 3 said the audit record it would require is unrecoverable if you did not write it at the time. Chapter 4 gave you two parameters and told you the real ones come from fitting. Chapter 5 told you a reproducible run can still be wrong. Chapter 6 said, in print, that this was the last time the book would defer it.

Fifteen deferrals come due here. They are listed in this chapter's dossier, and every one of them has a section.

The register, stated plainly, because it is the whole design of the chapter. This chapter teaches how to decide whether to believe a twin, and how to write the argument down so that somebody else can check it. **It teaches no estimation theory.** You will finish unable to derive a least-squares estimator, run a Bayesian calibration, or compute a sensitivity index. What you will be able to do is calibrate a two-parameter model by hand from a table, design a hold-out test, read residuals well enough to tell a broken model from a noisy sensor, set an alert threshold that a business sponsor can defend, notice when your data cannot tell two parameter sets apart, and write the five-part document that a regulator, a customer or an incident review will actually ask for.

What you are assumed to know. Everything so far. Especially: Chapter 1's value metric and its fidelity principle; Chapter 2's data-flow test and what changes when you close the loop; Chapter 3's seven components, in particular the store, the model registry and the actuation guard; Chapter 4's pot model with its parameters `k` and `g`, and its assumption ledger A1 to A5; Chapter 5's step size `h`, sampling interval `Ts`, and the difference between numerical error and being wrong about the world; Chapter 6's estimator gain and the `1/sqrt(N)` cost law.

The maths budget. Same as Chapter 6: no derivatives, no integrals, no matrices, no probability notation, no named distributions. The chapter uses exactly three operations on data – an average, a **spread**, and a square root – and computes the spread in full, by hand, once (Sec. 7.4.1). After that it is arithmetic and tables.

What this chapter deliberately does not cover. Estimation and optimisation theory: no least-squares derivation, no gradient descent, no Bayesian machinery, no sensitivity indices. Each is named where you would meet the word, and pointed at. Machine-learning risk in its own right – Chapter 8; Sec. 7.5.5 covers only what changes about the *argument* when the model is learned. Provenance plumbing – Chapter 10. Monitoring as a service – Chapter 11. Standards in depth – Chapter 13. The operational loop that re-validates a twin in production – Chapter 14; this chapter owes you the *criterion* for when belief expires, not the cron job that acts on it. The safety case – named in Sec. 7.7.6 as a different artifact with a different owner.

One warning about the data. The demonstrator's numbers in this chapter are constructed, not measured. There is a real plant-pot-and-pump rig behind this book, but

no calibration campaign has been run on it and this chapter invents none. Chapter 4 was equally explicit about its own numbers being illustrative. Everything about the *method* transfers; the specific values do not, and Sec. 7.12's last exercise is to go and get real ones.

Learning objectives

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

1. **Distinguish** verification, validation, calibration and uncertainty quantification, and **classify** a given activity, complaint or contract clause as belonging to one of them.
2. **Calibrate** a two-parameter model from a table of observations, **report** each parameter as a value with a spread, and **detect** when a body of data cannot tell two parameter sets apart.
3. **Design** a hold-out test, **read** its residuals to separate bias from noise, and **decide** whether an observed bias justifies adding a parameter.
4. **Derive** an alert threshold and the smallest fault it can catch from the model-reality gap and Chapter 1's value metric, and **separate** that gap from Chapter 5's numerical error.
5. **Write** a credibility argument with its five required parts, including a validity envelope and the conditions under which the argument expires.

7.1 Trust is not a property of a model

The chapter title asks the wrong question, deliberately, because it is the question you will be asked. "Should you trust the twin?" has no answer. It is the same shape as "is this code correct?" – correct *for what*, against *which specification*, under *which inputs*.

7.1.1 The question that does have an answer

Replace it with this one:

> **May this twin's output be used to make this decision, in these conditions, given this evidence?**

Four qualifications, and dropping any of them turns the question back into one nobody can answer.

This twin's output. Not "the twin". A twin computes several things. The demonstrator's fault detector compares an expected watering step against an observed one; its what-if service predicts a reading twelve hours ahead. Those are different claims with different evidence, and one can be sound while the other is not. In the literature this is put as the requirement that a measure of confidence be provided for the *outputs and predictions*, which in turn requires the sources of uncertainty in them to have been identified [1]. Outputs, plural.

This decision. Chapter 1's value metric decides how good is good enough, and Sec. 1.8.6 already derived a fidelity requirement from it. A model accurate to plus or minus 30 reading units is useless for scheduling doses to a target moisture and entirely adequate for noticing that a pump did not run. Same model, same accuracy, two verdicts, because two decisions.

These conditions. Chapter 4 Sec. 4.7 called this the validity envelope and was explicit that stating one is not establishing one. Establishing it is Sec. 7.5.3's job.

This evidence. Written down, checkable by somebody who was not there. Sec. 7.7 is the format.

7.1.2 How much evidence you owe depends on what the twin may do

This is the single most useful principle in the chapter, and it comes straight out of Chapter 1's ladder of value patterns.

Chapter 1 pattern	What a wrong answer costs	Evidence typically owed
Monitor	A misleading number on a screen; a human notices	Verification, plus a sanity check against measurements
Diagnose	A false alarm, or a missed fault	Calibration, a hold-out test, a stated threshold and its miss rate
Predict	A decision made early on a bad forecast	All of the above, plus a stated range on the prediction and an envelope
Decide (advisory)	A human acts on a bad recommendation	All of the above, plus the record of what the twin believed when it advised
Decide (closed loop)	The twin acts. Chapter 2 Sec. 2.8's world	All of the above, plus fail-safe behaviour, runtime envelope checks, and a safety argument that is not this document

The formal name for reading a table like that is **risk-informed credibility**: the depth of the assessment is set by the consequence of being wrong, not by how much rigour the team enjoys. It is the organising idea of the American Society of Mechanical Engineers (ASME) standard VV-40, which the medical-device world uses for exactly this purpose [2], and Chapter 13 is where standards get their own treatment. What matters here is the direction of the inference. **You do not decide how much V&V to do and then see what the twin may be allowed to do. You decide what the twin is allowed to do, and that tells you the bill.**

The reverse mistake is the expensive one. A team that validates to aerospace depth a twin whose only job is to colour a dashboard has spent six months buying nothing. A team that closes the loop on the strength of "the model looked good in the notebook" has bought a liability.

7.1.3 Why this is a software engineer's problem

Three reasons, and none of them are that you will fit the parameters.

The evidence is mostly infrastructure. Every test in this chapter needs data the model has not seen, held with its timestamps, its units, its sensor identity and its version

metadata intact. That is Chapter 3's store and Chapter 10's data engineering. A modelling expert who asks for two weeks of held-out readings and gets a spreadsheet with the times rounded to the hour cannot do their job, and will not always tell you why.

The claim expires, and something has to notice. A twin is not validated once. The literature is explicit that validation of a twin differs from classical simulation validation exactly here: in classical practice a model is calibrated, then validated, then considered finished, whereas a twin's model must be *continually* re-checked against a system that changes, with recalibration triggered when it no longer matches [3]. The thing that notices is a monitor you build.

The argument is a deliverable with a filename. Chapter 2 Sec. 2.11 asked you to write a contract clause and pointed out that a term nobody can check is not a term. Sec. 7.7.5 is the clause. Somebody has to own the document, version it, and fail a release when it is stale. In practice that is the same person who owns the model registry, and that is you.

7.2 Four questions hiding in one word

People say "we validated the model" to mean at least four different things. Separating them is worth doing once, carefully, because the four have different inputs, different owners and a **mandatory ordering** that teams get wrong.

> **Verification** – did we build the model *right*? Does the code implement the equations somebody intended, with solver settings adequate to the job, and does the plumbing deliver the numbers unaltered? An internal question. **You can answer it without measuring the physical twin at all.**
>
> **Calibration** – what values should the parameters have? Choosing numbers so that the model's output matches observed data. Also called parameter estimation [4].
>
> **Validation** – did we build the *right* model? Does it match the part of reality the decision depends on? An external question. **Unanswerable without measurements the model has not already been fitted to.**
>
> **Uncertainty Quantification (UQ)** – how wrong might the answer be, and from which causes?

The standard shorthand bundles all four as **Verification, Validation and Uncertainty Quantification (VVUQ)** – Uncertainty Quantification (UQ) on its own when the last of them is meant – and it is the term you will meet in procurement documents and standards gap analyses – ISO 23247, the manufacturing digital-twin framework standard, is criticised in exactly these words for not covering it [5]. The formal definitions are old and stable: verification is the check that the programmed model is a correct implementation of the conceptual model, performed before any parameter is fitted [3].

7.2.1 A table you can hang on a wall

	Verification	Calibration	Validation	UQ
Question	Did we build it right?	What are the numbers?	Did we build the right thing?	How wrong might it be?
Compares	Code against intent	Model against fitting data	Model against **unseen** data	Answer against its own error budget
Needs measurements?	No	Yes	Yes, and different ones	Yes, from the two before it
Typical owner	You	Modelling expert	Both, plus whoever owns the decision	Modelling expert, reported by you
Fails like	Wrong sign, wrong units, step too coarse	Parameters that fit nothing, or fit anything	Good fit, wrong predictions	A confident number with no range on it
Fixed by	Reading code, refining the step	More or better data	A different model, or a smaller claim	Honesty

7.2.2 The ordering trap, which is the one to remember

The four have an order: **verify, then calibrate, then validate, then quantify.** Figure 7.1 draws the order with the two common breaks marked, because each break has a distinct symptom rather than a general smell.

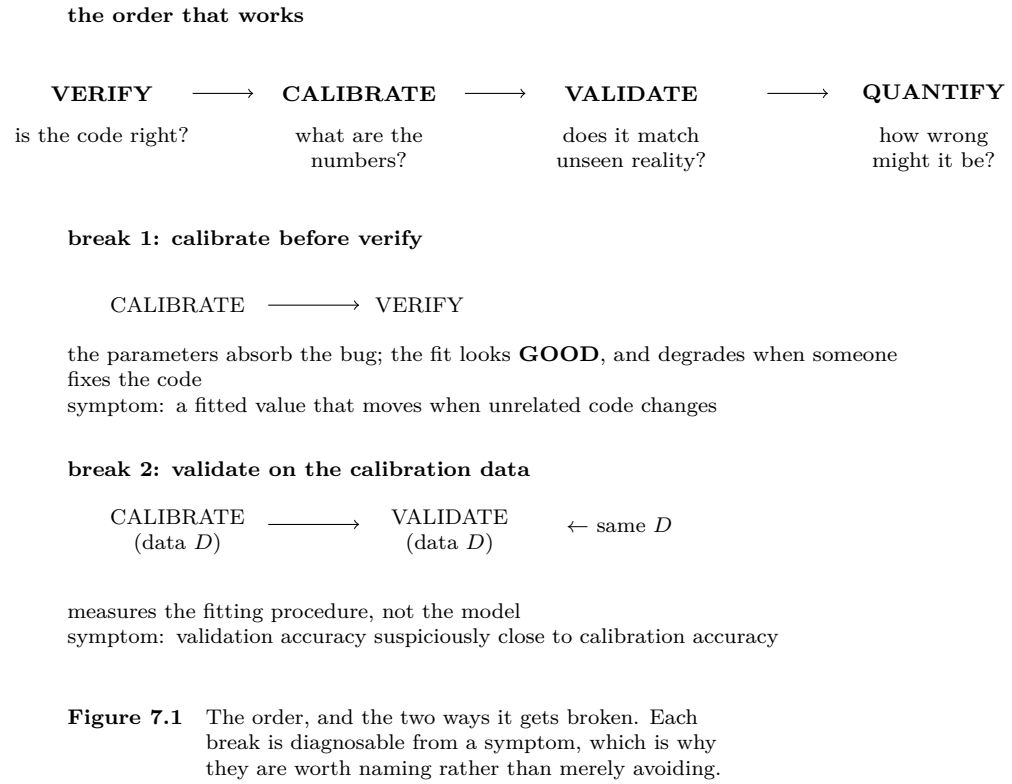

Figure 7.1 The order, and the two ways it gets broken. Each break is diagnosable from a symptom, which is why they are worth naming rather than merely avoiding.

Two ways teams break it, both common enough to have symptoms.

Calibrating before verifying. The parameters absorb the bug. Suppose the code

applies the watering dose at the end of the step instead of the start – Chapter 5 Sec. 5.2.4 showed exactly that ambiguity, and nothing warned us. Fit `k` and `g` against data with that bug in place and the fitted values will quietly compensate for it. The fit will look *good*. Then somebody fixes the bug, the fit degrades, and the team concludes the fix was wrong. **Symptom: a parameter whose fitted value changes materially when a piece of code that should not affect it is changed.**

Validating on the calibration data. The model was chosen to match those numbers. Testing it on them measures the fitting procedure, not the model. This is the train/test split under a different name, and every engineer who has trained a classifier already knows it – the failure is not ignorance, it is that in modelling projects the two datasets are often the same expensive measurement campaign, and the temptation to use all of it for fitting is real. **Symptom: a validation report whose accuracy figure is suspiciously similar to the calibration report's.**

There is a third, subtler break, and Sec. 7.4.5 walks into it on purpose so that you can watch it happen: **using the hold-out data to fix the model.** The moment you do, it is no longer hold-out data, and you owe a fresh set.

7.3 Verification: the part that is ordinary software engineering

Good news first. Verification is the quarter of this chapter that needs no statistics, no modelling expert, and no measurement campaign. It is code review, tests, and version discipline, and you are already good at it.

7.3.1 Three layers, and they fail differently

Chapter 6 Sec. 6.1 stacked method, algorithm, solver and simulator. For verification the useful stack is shorter.

Layer 1: the model as written. Does the code compute the equations the assumption ledger describes? The demonstrator's model is nine lines and fits in a code review, which Chapter 4 called the appeal of physics-based models and which is also its verification story. Things to check by reading:

- **Units and their scale.** `g` is reading units per *100* millilitres. Code that multiplies by the dose in millilitres directly is wrong by a factor of a hundred, and a factor of a hundred shows up in the first plot anybody draws. Code that is wrong by a factor of 1.2 does not.
- **Signs.** Drying subtracts, watering adds. A sign error in a model that is symmetric enough will still produce a plausible curve.
- **Event placement.** Where in the step does a dose land? Chapter 5 Sec. 5.2.4 put it forty-five minutes early and said nothing.
- **Boundaries.** What does the model do at reading 1023, the top of the sensor's range? Assumption A2 says it stops being affine there. Does the code know?

Layer 2: the solver and its settings. Chapter 5's `h`, or an adaptive solver's tolerance. Sec. 7.3.2 is entirely about this layer, because it is where the confusion between two different errors lives.

Layer 3: the plumbing. Chapter 3's connector, store and runner, between the sensor and the model's input. Resampling, unit conversion, timezone handling, gap filling, the interpolation that Chapter 5 Sec. 5.2.4 warned was an assumption. This layer produces the most verification defects in practice and gets the least attention, because it does not look like modelling. A twin that reads a moisture value from a store that silently forward-fills a two-hour sensor outage is being lied to by its own infrastructure, and no amount of validation will find it – the model is fine.

> **A rule worth adopting.** Verify each layer against the layer above it with the layer below held fixed. Test the model with synthetic inputs you constructed by hand. Test the plumbing with a constant signal whose value you know. Do not debug all three at once against live data, which is what "the twin's numbers look wrong" usually degenerates into.

7.3.2 Numerical error is not the model-reality gap

Chapter 5 promised this section three times and Chapter 6 promised it once. Here it is, and it is the most confused pair of quantities in the subject.

There are two entirely different ways the number on the screen can differ from the truth.

- **Numerical error** – the gap between what the model *says* and what the solver *computed*. Caused by step size, tolerance, event placement. Belongs to verification. **Measurable without any sensor.**
- **The model-reality gap** – the gap between what the model says and what the world does. Caused by assumptions A1 to A5 being imperfect, by parameters being slightly off, by the world containing things the model omits. Belongs to validation. **Not measurable without sensors.**

They are not comparable in kind and they are routinely added together and called "the error".

The check that separates them, and it costs one afternoon. Take the scenario you intend to validate on. Run it at your chosen step size. Then run it again at half that step, and again at half of that, and watch what happens to the answer. Chapter 5 Sec. 5.4.2 ran exactly this experiment on the exponential form of the pot model – reading falls toward a floor of 400, time constant 48 hours, starting at 640, over 24 hours of drying with no watering. The exact answer, from the mathematics rather than the solver, is 545.6. Extending Chapter 5's table downward:

`h` (hours)	steps	result at t = 24 h	numerical error
24	1	520.0	25.6
12	2	535.0	10.6
6	4	540.7	4.9
3	8	543.2	2.4
1.5	16	544.4	1.2
0.75	32	545.0	0.6
0.375	64	545.3	0.3

Each halving of `h` halves the error, which is Chapter 5's first-order rule behaving as

advertised. **Now the point.** Section 7.4.6 will measure this model's gap against reality, on data it has never seen, and get about **2 reading units**. Read that number back into the table:

- At `h` = 24 the numerical error is 25.6, twelve times the model-reality gap. A validation run at that step size measures your step size. It says nothing whatever about your model.
- At `h` = 3 the numerical error is 2.4, slightly *larger* than the gap you are trying to measure. Still useless, and this is the dangerous row, because 2.4 looks small.
- At `h` = 0.375 the numerical error is 0.3, about a seventh of the gap. Now a validation result is about the model.

  The rule. Before validating anything, drive the numerical error to a small fraction of the gap you expect to measure – a fifth is a defensible convention. Then say in the credibility argument which step size the validation was run at, because the result is only valid for runs at that setting or finer.

Two practical notes. First, run the convergence check **on the scenario you will validate on**, not on a convenient one; the table above is a 24-hour dry-down, and a scenario with watering events in it will behave differently because the events interact with the step boundaries. Second, if halving the step keeps changing the answer by the same amount instead of half as much, stop and get help: that is not convergence, and the usual causes are an event landing on a step boundary or the instability Chapter 5 Sec. 5.4.4 described.

7.3.3 Golden trajectories, and comparing model versions

Chapter 5 Sec. 5.6 said that Chapter 7 would want to compare model versions against the same scenario. This is that.

A **golden trajectory** is a recorded scenario – initial state, inputs, horizon, step size, solver identity and version – together with the output the current model produces for it, checked into version control beside the model. It is a snapshot test, and everything you know about snapshot tests applies, including that they rot.

Three things it buys you, in increasing order of value.

A regression test. A refactor of the model code that changes no behaviour changes no trajectory. This catches the sign error, the unit error, and the accidental step-size default change.

A diff between model versions. When the modelling expert delivers version 3, run version 2 and version 3 against the same golden scenarios and diff the outputs. You now have a sentence for the release notes: "version 3 predicts evening readings 4 units lower on average across the twelve regression scenarios, and differs by more than 10 units only in the saturation scenario." Without golden trajectories that sentence does not exist, and model upgrades happen on trust.

A test for the change nobody declared. The most common way a twin degrades is that something changed which nobody thought was part of the model – a library version, a default tolerance, a timezone database. A golden trajectory that starts failing after a dependency bump has told you something no other test would.

The discipline that makes this work, and it is not optional. A golden trajectory whose recorded inputs are incomplete is worse than none, because it fails intermittently and gets deleted. Chapter 5 Sec. 5.6 listed what a run needs to be reproducible; the golden trajectory must carry all of it, including the solver version, and must be *handed* its inputs rather than fetching them. Chapter 5 named a runner that fetches instead of being handed as the commonest cause of irreproducibility, and it is also the commonest cause of a flaky golden test.

An automated approach to this is a live research and practice area rather than a solved one. Work on systematic digital-twin testing for industrial processes builds an architecture around exactly this idea – snapshot creation plus a testing agent that runs online and detects performance shift – and reports that it accelerates the testing process and improves its reliability [6]. The multi-level testing frameworks being built for cyber-physical systems more generally [7], and the requirements-driven testing that carries a scenario from simulation through to the real system [8], are the same instinct applied at larger scale. Continuous-integration practice for cyber-physical systems is itself an uncomfortable fit, and an interview study of practitioners found the hardware-in-the-loop stages to be the awkward part [9] – a finding a software engineer should read as permission to expect this to be harder than it is for a web service.

7.3.4 Reproducibility is a precondition, not an achievement

Chapter 5 Sec. 5.6 argued for reproducibility and then said something that this chapter has to finish: **a seed makes a run repeatable, not correct.**

Reproducibility buys you the ability to *have* the argument. Without it you cannot compare two model versions, cannot re-run last Tuesday's alert during the incident review, and cannot demonstrate that the number in the credibility argument is the number the system produces. With it you have none of those answers yet – you merely have a system in which the questions are askable.

The failure this prevents is specific and worth naming, because it is the one that destroys a credibility argument in a meeting. Somebody asks why the twin raised an alert at 17:20 on the 14th. You re-run the scenario and get no alert. Now the argument is not about the model at all; it is about whether the system can be reasoned about, and you have lost. Chapter 3's audit trail and Sec. 7.7.3's record exist to prevent that specific conversation.

7.3.5 What to ask

- Which layer is this defect in – the model, the solver settings, or the plumbing? (If nobody can say, that is the answer.)
- What step size or tolerance was this result produced at, and what does the convergence table look like at half of it?
- Where are the golden trajectories, and when did one last fail?
- Can you re-run the run that produced this number, today, and get this number?

7.4 Calibration: getting the numbers out of the data

Chapter 4 gave the pot model two parameters, `k` and `g`, put illustrative values on them by reading two points off a chart, and said the real ones come from fitting. This section fits them.

> **Calibration.** Choosing values for a model's parameters so that its predictions match observed data. Also called parameter estimation. It finds values that optimise the model's accuracy about the future evolution of the modelled system [4].

Note what calibration is *not*. It does not change the model's structure, it does not test the model, and it cannot fix an assumption. A wrong model with well-fitted parameters is a wrong model that fits.

7.4.1 Fitting `k`, by hand, from six nights

Chapter 4 estimated `k` from one dry night: the reading fell from 640 to 592 over twelve hours, so `k` = 48 / 12 = 4 reading units per hour. One observation, one number, no idea how good it is.

A calibration campaign is that, repeated. Take six overnight windows with no watering, each twelve hours long, from the first week of logging. The drops, in reading units:

Night	Reading at start	Reading at end	Drop	Rate (drop / 12)
1	640	592	48	4.00
2	631	589	42	3.50
3	637	592	45	3.75
4	628	589	39	3.25
5	641	597	44	3.67
6	635	589	46	3.83

The estimate. Average the drops: (48 + 42 + 45 + 39 + 44 + 46) / 6 = 264 / 6 = 44.0 reading units per night. Divide by 12 hours:

```
k = 44.0 / 12 = 3.67 reading units per hour
```

The spread, computed once in full. A single number is not a calibration result. How much do the nights disagree? Take each drop's deviation from 44.0, square it, add the squares, divide by one less than the count, and take the square root. This chapter calls the result the **spread**; the rest of the world calls it the standard deviation. (Dividing by the count rather than one less than it is also common and changes the answer very little; nothing in this chapter turns on which you use, but be consistent.)

```
deviations:   +4    -2    +1    -5     0    +2
squares:      16     4     1    25     0     4      sum = 50
divide by 5:  50 / 5 = 10
spread:       square root of 10 = 3.16 reading units per night
```

Divide by 12 hours to put it in the same units as `k`:

```
spread of k = 3.16 / 12 = 0.26 reading units per hour
```

Report both. `k` = 3.67 plus or minus 0.26 reading units per hour. The 0.26 is not a decoration: Sec. 7.5.2 spends it, and a parameter delivered without one cannot be spent at all.

One more number, free from Chapter 6. The spread of 0.26 describes how much *one night* differs from the average. The uncertainty in the *average itself* is smaller, and shrinks with the number of nights by Chapter 6 Sec. 6.7.3's law: divide by the square root of the count.

```
uncertainty in the average = 0.26 / square root of 6 = 0.26 / 2.45 = 0.11
```

So six nights pin the average nightly rate to about plus or minus 0.11. To halve that you would need twenty-four nights, which is Chapter 6's cost law arriving in a place you did not expect it. **Note that these are two different quantities and confusing them is a real error:** 0.26 is how much a night varies, 0.11 is how well you know the average. Predicting one night's drying uses 0.26. Arguing about whether `k` has changed since last month uses 0.11.

7.4.2 Fitting g, and why the two campaigns look different

`g` is the step in the reading produced by 100 millilitres. It needs a different experiment: a dose, and readings tight around it. Five doses of 100 ml, each measured as the reading ten minutes after minus the reading just before:

Dose	Step observed
1	47
2	50
3	44
4	49
5	45

Average: 235 / 5 = **47.0 reading units per 100 ml**.

Deviations 0, +3, -3, +2, -2; squares 0, 9, 9, 4, 4, sum 26; divide by 4 gives 6.5; square root gives a spread of **2.55**.

```
g = 47.0 plus or minus 2.55 reading units per 100 ml
```

Chapter 4's by-eye figure was 48. The fitted figure is 47.0. That gap of 1.0 is smaller than the spread, so nothing dramatic has happened – but Sec. 7.5.2 will show that 1.0 is not free either.

Why the two campaigns differ, and this is the transferable part. Fitting `k` needed *long* windows with *no* doses. Fitting `g` needed *short* windows *around* doses. Neither experiment can fit the other parameter. That is not a coincidence, and Sec. 7.4.4 is about what happens when you ignore it.

7.4.3 The cost function is a choice, and it is your business

Somewhere inside every fitting procedure is a rule for what "matches the data" means. Above, it was: make the average residual zero. The standard choice is to minimise the sum

of the *squared* residuals, which is what least squares means and what most tooling does by default [4]. For models where that will not work – nonlinear ones – the machinery escalates to iterative search, and in digital-twin practice to design-space exploration techniques including genetic algorithms and particle swarm optimisation, all of them defining an objective function that measures how well the model aligns with the calibration data [4].

You will not choose the algorithm. You should know that the objective is a choice, because it encodes what you care about, and three consequences reach your code:

- **Squared residuals punish large misses hard.** One badly wrong day pulls the fit further than five slightly wrong days. If your data contains a sensor dropout that reads 0, the fit will contort itself around it. **Ask what happened to the outliers.**
- **The objective may not be the thing the twin is for.** The demonstrator's value comes from detecting a *missing dose* – a large, rare, sudden event. A fit that minimises average error across all hours is optimising for the ordinary hours. It is usually still the right choice, but somebody should have said so out loud.
- **Weighting is available and rarely used.** If evening readings matter more than morning ones because the evening dose is the risky one, the fit can be told. It will not guess.

7.4.4 Identifiability: when the data cannot tell two answers apart

Chapter 2 Sec. 2.3.3 said that answering "is the model faithful enough?" takes Chapter 7 *and a measurement campaign.* This is why the campaign is not optional, and it is the most expensive mistake in this chapter.

Suppose that instead of the two campaigns above, you had only what the shipped logger already records: one reading at 09:00 and one at 21:00 each day, with a single 100 ml dose at 17:05 in between. Over those twelve hours the model says the reading falls by `k` times 12 and rises by `g` times 1.0. The net change is:

```
net change over the day = g - 12k
```

Suppose the observed net change averages +3.0 reading units. Now look at what fits:

Candidate	`k`	`g`	Predicted net change
A	3.67	47.0	47.0 - 44.0 = 3.0
B	3.00	39.0	39.0 - 36.0 = 3.0
C	4.50	57.0	57.0 - 54.0 = 3.0

All three fit the data exactly equally well. There are infinitely many more. The daily data cannot distinguish them, and no cleverer fitting algorithm will help, because the information is not in the data. The parameter pair is **unidentifiable** from this body of observations.

> **Identifiable.** A parameter set is identifiable from a body of data if no *other* parameter set fits that data about as well.

Why this is not an academic worry. The three candidates agree on the daily total and disagree violently about the thing the twin exists to compute. The expected step from the evening's 120 ml dose is `g` times 1.2:

Candidate	Expected step from 120 ml
A	47.0 x 1.2 = 56.4
B	39.0 x 1.2 = 46.8
C	57.0 x 1.2 = 68.4

Candidates B and C differ by 21.6 reading units. Section 7.5.2 will set the fault-alert threshold at 9 units. A twin calibrated on daily data has a one-in-however-many chance of landing on a parameter pair that either alarms on every healthy dose or never alarms at all – **and the fit will look perfect either way.**

The fix is experimental design, not statistics. You separate the parameters by taking observations in which only one of them acts: a dry night with no dose isolates `k`; a reading pair tight around a dose isolates `g`. That is precisely what Sec. 7.4.1 and Sec. 7.4.2 did, and now you know why they were two campaigns instead of one.

Three symptoms you can spot without doing any maths.

1. The fitted parameters change a lot when you refit on a different week, while the fit quality stays the same.
2. Two parameters move together whenever either is changed – raise one and the fit recovers by raising the other.
3. The modelling expert says the fit is "flat" or "the optimiser wanders". That is the same fact in their vocabulary.

What to do about it. Ask for an experiment that moves one parameter's effect without the other's, and be prepared for the answer that it requires access to the physical twin you were not planning to interrupt. This is where model-updating practice in the field lives: structural work fits finite-element parameters against dedicated experimental measurement campaigns rather than against operational logs, precisely because operational data does not excite the modes that separate the parameters [10].

7.4.5 Hold-out: the test that finds what the fit hid

The parameters are fitted. Nothing yet suggests they predict anything. To find out, use data the fit never saw.

> **Hold-out.** Data withheld from calibration and used only to test the calibrated model. The train/test split, under its modelling name.

Figure 7.2 shows the discipline and the way it decays, which is the third break Sec. 7.2.2 promised you would watch happen.

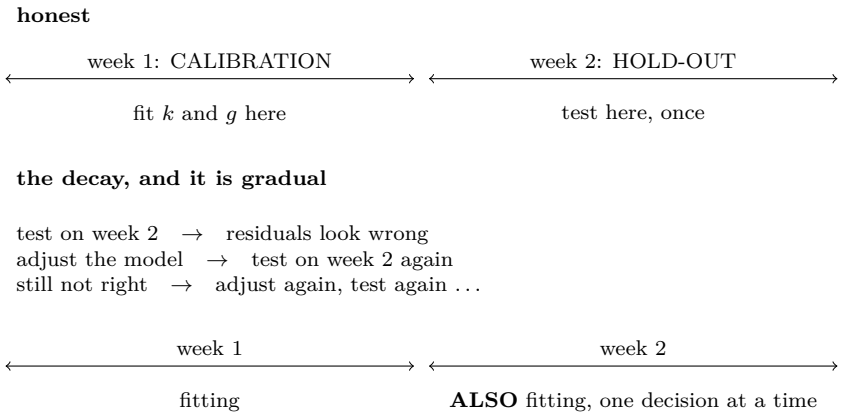

Figure 7.2 Hold-out decays by use. Every look that changes the model spends some of it, and the spending is silent – which is why the honest move is to write down in advance what the hold-out is allowed to decide.

The figure's last line is the practical defence. Decide before you look whether the hold-out is being used to *accept* the model or to *improve* it; only the first leaves it intact.

Set it up honestly. Week 1 was the calibration week. Week 2 is the hold-out. Note something uncomfortable before starting: `k` was fitted on *nights*, and the prediction we care about spans a *day* – the 7.8-hour window from the 09:16 dose to the 17:05 dose. Assumption A1b in Chapter 4's ledger anticipated exactly this, and warned that a single `k` would over-predict overnight drying and under-predict daytime drying. Let us see if it does.

For each day of week 2, take the reading just after the morning dose, predict the reading just before the evening dose using a drop of 3.67 x 7.8 = **28.6** reading units, and compare with what the sensor actually recorded. The residual is Chapter 1's quantity: measured minus predicted.

Day	Morning reading	Predicted evening	Measured evening	Residual
1	638	609.4	605	-4.4
2	642	613.4	606	-7.4
3	635	606.4	603	-3.4
4	640	611.4	606	-5.4
5	644	615.4	612	-3.4

Read the residuals, not the model. Two numbers matter and they say different things.

The average residual is the bias. (-4.4 - 7.4 - 3.4 - 5.4 - 3.4) / 5 = -24.0 / 5 = **-4.8**. Every single day the pot is drier than predicted, by about five units.

The spread of the residuals is the noise. Deviations from -4.8 are +0.4, -2.6, +1.4, -0.6, +1.4; squares 0.16, 6.76, 1.96, 0.36, 1.96, sum 11.2; divided by 4 gives 2.8; square root gives **1.67**.

Bias and spread are the two things a residual set tells you, and confusing them wastes weeks. A bias of -4.8 with a spread of 1.67 is a model that is wrong in a specific, repeatable direction – nearly three times the noise. That is *good news*, because a repeatable error is fixable. A bias of 0.0 with a spread of 4.8 would be a model that is right on average and useless in particular, which is much harder to fix.

Compare with what would have happened had the team shipped Chapter 4's by-eye `k` = 4.00. The predicted drop would be 4.00 x 7.8 = 31.2 instead of 28.6, so every prediction sits 2.6 lower and every residual moves 2.6 higher: the bias becomes -4.8 + 2.6 = **-2.2**. Smaller. **The by-eye parameter happens to fit the daytime data better than the carefully fitted one, because night 1 was the driest night in the calibration set and its rate accidentally sat closer to the daytime rate.** That is not an argument for guessing. It is an argument for reading Sec. 7.4.4 again: a parameter fitted on the wrong experiment can be beaten by luck.

7.4.6 When an extra parameter is earned, and what it costs

The bias has a named cause sitting in the assumption ledger. A1b: one drying rate for day and night. Split it.

The daytime rate falls out of the hold-out week's own numbers. The measured drops over the 7.8-hour windows were 33, 36, 32, 34 and 32, averaging 167 / 5 = 33.4, so:

```
k_day = 33.4 / 7.8 = 4.28 reading units per hour
```

with a spread of 1.67 / 7.8 = 0.21. The model now has three parameters: `k_night` = 3.67, `k_day` = 4.28, `g` = 47.0.

And the hold-out week is now burned. It was used to *fit* `k_day`. It can no longer test anything, and any accuracy figure computed on it is a calibration figure wearing a validation figure's clothes. This is Sec. 7.2.2's third ordering trap, and it is a comfortable mistake to make, because fixing the model with the data that revealed the flaw feels like diligence.

So: week 3, untouched, tests the three-parameter model. Predicted drop is 4.28 x 7.8 = **33.4**.

Day	Morning reading	Predicted evening	Measured evening	Residual
1	639	605.6	607	+1.4
2	641	607.6	605	-2.6
3	636	602.6	604	+1.4
4	643	609.6	611	+1.4
5	637	603.6	602	-1.6

Bias: (1.4 - 2.6 + 1.4 + 1.4 - 1.6) / 5 = 0.0 / 5 = **0.0**. Spread: deviations are the residuals themselves; squares 1.96, 6.76, 1.96, 1.96, 2.56, sum 15.2; divided by 4 gives 3.8; square root gives **1.95**, call it **2.0**.

The bias is gone and the spread is unchanged. That is what a well-earned parameter looks like: it removes a systematic error and does not merely absorb noise. Record the result, because the whole rest of the chapter spends it:

> **The model-reality gap for the demonstrator's day-ahead prediction is about 2.0 reading units, with no detectable bias, on week 3 data, at step size 0.375 hours.**

Now the discipline that stops this becoming a habit. Every parameter you add improves the fit. That is arithmetic, not evidence. A parameter is earned only if:

1. It corresponds to a *named* row of the assumption ledger, so you can say what physical thing it represents. `k_day` does; a fourth-order polynomial correction does not.
2. Removing the bias survives on data neither the original fit nor the fix has seen. Week 3 did that.
3. Somebody will maintain it. Three parameters means three things to refit when the plant is repotted, and Chapter 14 owns that bill.

Fail any of the three and you are **overfitting**: buying fit with generality. The modelling literature names it in the same breath as choosing network sizes, and the guard is the same one you already use, which is holding data back [11].

7.4.7 Calibration is not a one-off

Chapter 4's assumption A4 said parameters are constant over weeks, and that after repotting or rapid growth every prediction drifts. That makes recalibration a scheduled activity, not an incident.

This is one of the genuine differences between validating a simulation model and validating a twin. In classical modelling and simulation practice the model is calibrated, validated once, and then generally assumed finished; for a twin, validation has to be **continual**, with recalibration triggered when the model stops matching the world it is attached to [3]. The reference twin architectures build this in: the incubator case study wires a calibration service into the twin itself, with the monitoring phase watching prediction error and the planning phase invoking calibration to estimate new parameters [12].

Two things belong in your design because of this, and both are cheap now and expensive later.

A recalibration trigger. Not a calendar entry – a condition on the residuals you are already computing. Sec. 7.7.4 states one.

A parameter history. Every set of fitted parameters, with the data window it was fitted on, who ran it, and the resulting hold-out figures. Chapter 3's model registry is where this lives. Without it, "the twin has been getting worse for a month" is unanswerable, and with it the answer is often a one-line diff.

7.4.8 What to ask

- Which experiment fitted each parameter, and can it fit that parameter on its own?
- What is the spread on each fitted value?
- What did the objective function do with the outliers?

- Which data is held out, and has it stayed held out?
- What triggers a refit, and where is the history?

7.5 Validation: does it match a reality it has not seen

Calibration made the model agree with data. Validation asks whether it agrees with the world. The distinction only becomes real when the data is different, which is why Sec. 7.4.5 exists before this section.

7.5.1 The rule, and the three ways teams get around it

Validation data must be data the model has not been fitted to. That is the whole rule. The ways around it are:

Fitting on everything and validating on a subset of it. Already covered (Sec. 7.2.2).

Fitting on everything, then fixing the model using the validation results. Covered, and demonstrated on purpose (Sec. 7.4.6).

Validating on data from a different period that is not actually different. Two weeks of the same fortnight, same weather, same plant, same pot, does not test assumption A4 at all. It tests noise. A hold-out that shares every condition with the calibration set validates the fit, not the model. Where you can, hold out a period that differs in something the envelope claims not to care about.

7.5.2 How close is close enough? Derive it from the decision

Here is where the chapter cashes Chapter 1's fidelity principle: *fidelity is a requirement derived from the value metric, not a virtue pursued for itself.*

The demonstrator's decision is Chapter 1's: raise an alert when a scheduled dose did not deliver water. The twin computes an expected step and compares it with the observed one.

Step 1 – what the model predicts. Evening dose is 120 ml, so the expected step is g x 1.2 = 47.0 x 1.2 = **56.4** reading units.

Step 2 – how much a healthy dose varies. The calibration campaign's 100 ml doses had a spread of 2.55. Scaled to 120 ml that is 2.55 x 1.2 = 3.06, call it **3.0** reading units.

Step 3 – choose a threshold in units of that spread. Take three spreads: alert when the observed step falls more than 3 x 3.0 = **9.0** units below expected, that is, whenever the observed step is below 56.4 - 9.0 = **47.4**.

Three spreads is a convention, not a derivation. What it buys is that ordinary variation rarely crosses it. What it costs is stated next, and stating that cost is the part teams skip.

Step 4 – state the smallest fault it can catch. A shortfall of 9.0 units out of an expected 56.4 is 9.0 / 56.4 = **16 per cent**. So:

Fault	Water delivered	Observed step	Shortfall	In spreads	Caught?
Pump did not run	0 ml	about 0	56.4	18.8	Yes, unmistakably
Dripper half blocked	60 ml	28.2	28.2	9.4	Yes
20 per cent shortfall	96 ml	45.1	11.3	3.8	Yes, just
10 per cent shortfall	108 ml	50.8	5.6	1.9	**No**

Step 5 – go back to Chapter 1 and ask whether the blind spot matters. The value metric was experiment-weeks lost to undetected watering faults. A 10 per cent shortfall does not lose an experiment week; the plant does not notice and neither does the researcher. **The blind spot is acceptable, and you can now say so in a sentence a sponsor can check**: "this twin detects watering shortfalls of 16 per cent or worse; smaller shortfalls are not detected and are not expected to affect plant health."

That sentence is worth more than any accuracy percentage, because it is in the units of the decision rather than the units of the model.

Step 6 – notice what the calibration bias would have cost. Suppose the team had shipped Chapter 4's by-eye $g = 48$. Expected step 57.6, threshold at 57.6 - 9.0 = 48.6. But healthy doses actually centre on 56.4, so the alarm now fires whenever a dose falls 56.4 - 48.6 = 7.8 units short, which is 2.6 spreads instead of 3.0. **A 1.0-unit calibration error in g – well inside its own spread, and invisible in any fit-quality number – moved the alarm threshold by 13 per cent and will produce false alarms nobody can explain.** That is what Sec. 7.4.2 meant by saying the gap of 1.0 was not free.

7.5.3 Establishing the validity envelope

Chapter 4 Sec. 4.7 defined the validity envelope and Sec. 4.7.2 said plainly that stating one is not establishing one. Establishing it means turning each row of the assumption ledger into a *tested boundary* or an *admitted limitation*. There is no third option, and the value of the exercise is that it forces the admission.

Ledger row	Assumption	How it becomes an envelope clause	Established how?
A1	Constant outflow rate over the interval	Valid for prediction horizons up to 12 hours	Tested: week 3 hold-out used 7.8-hour windows; 24-hour windows not tested
A1b	One rate day and night	Superseded by two rates (Sec. 7.4.6)	Tested and fixed

Ledger row	Assumption	How it becomes an envelope clause	Established how?
A2	Reading affine in water content	Valid for readings between 350 and 900	**Not tested.** Sensor datasheet claim, adopted as a stated limitation and enforced as a runtime check
A3	Delivered water reaches sensed soil	The twin cannot distinguish pump failure from a displaced dripper	**Cannot be tested from data.** Admitted limitation; Chapter 4 already argued the value metric tolerates it
A4	Parameters constant over weeks	Valid for 4 weeks after a calibration; expires then	**Partly tested.** Three consecutive weeks showed no drift; 4 weeks is an extrapolation of one week
A5	One number describes the pot's water	No claim is made about surface pooling or layered drying	Admitted limitation

Three things to notice, because they generalise past this pot.

Most rows end as admitted limitations, and that is a healthy document. An envelope in which everything was tested means either an unusually generous measurement budget or an unusually optimistic author. The list of limitations is what makes the tested claims believable.

The envelope must be enforced in code, not just written down. Chapter 4 Sec. 4.7 already made this your job: if the model is valid between 350 and 900, the service that calls it refuses, loudly, outside that band. This is the difference between an envelope and a wish. The hazard is silent failure – a model that returns 608.8 when the truth is 400 is far worse than one that crashes, because 608.8 is plausible and nothing downstream can tell. The literature makes the same point at the level of the whole twin: models are limited by their assumptions, by the data available to build and validate them, and by what the modellers knew, and those factors bound the domain of validity of the twin itself [1].

The envelope has a clock. Row A4 gives the credibility argument an expiry date, which Sec. 7.7.2 requires it to state.

7.5.4 What would count against it? The falsifiability question

Chapter 4 Sec. 4.9 gave you five questions to ask a modelling expert and flagged one answer as worrying: no answer at all to "what observation would tell you this model is

wrong?". A model that no observation could count against cannot be validated – there is nothing for validation to do.

Made concrete for the pot, the answers are good ones:

Claim	An observation that would falsify it
Drying is at a constant rate over 12 hours	A dry-down curve that visibly bends within a 12-hour window
The daytime rate exceeds the night rate	Daytime windows whose fitted rate is not above 3.67 plus its spread
A 120 ml dose steps the reading by 56.4	Healthy doses whose steps average materially away from 56.4
The parameters hold for 4 weeks	Residual bias growing steadily across weeks 4 to 8

Each row is a test somebody could run, and rows 3 and 4 are tests the running twin can perform on itself forever, which is what Sec. 7.7.4 turns into monitors.

Use this question as a design tool, not a debating tactic. If a proposed model produces a table with empty right-hand cells, the problem is not the modelling expert – it is that the model as specified makes no checkable claim, and the time to discover that is before the measurement campaign is budgeted.

7.5.5 Validating a model whose parameters mean nothing

Chapter 4 Sec. 4.4.3 said a learned model's coefficients afford no inspection, and that Chapter 7 would need something to check. Chapter 4 Sec. 4.6 said the position on the white-to-black-box scale decides *what kind of argument* you can make. Chapter 6 Sec. 6.5.1 reported that surrogates can fail unexpectedly on inputs unlike anything in training, and said flatly that this limit is why Chapter 7 exists. All three arrive here.

What changes. Nothing about the *procedure* – calibrate, hold out, read the residuals, state an envelope. Everything about the *argument*, because three of the moves used above are unavailable:

Move	Physics-based pot model	Learned model
Inspect a parameter for plausibility	"Is 4.28 units per hour reasonable for this plant?" Somebody can answer	No coefficient means anything on its own
Attribute a bias to a named assumption	A1b, immediately	Nothing to attribute it to; the fix is more data or a different architecture
State an envelope from the ledger	Six rows, mostly free	The envelope must be derived from the **coverage of the training data**, which somebody has to characterise

Move	Physics-based pot model	Learned model
Predict the failure mode	Drifts in a direction the assumptions predict	Confident output in a region with no training data, with no signal that anything is wrong

So the argument shifts entirely onto the data. For a learned model or a surrogate, the credibility argument's evidence section is mostly about inputs: what range of conditions the training data covered, how the hold-out was constructed, and what the model does when it is handed something outside that range. The last of those is the part teams skip and the part that bites – the recognised danger is precisely unexpected failure on inputs unlike the training set, and the recognised consequence is reduced trust and limited real-world adoption [13].

Two practical mitigations, both of which are your build.

Input-coverage checks at the boundary. Before calling the model, check that the inputs resemble the training range on each dimension you can name. This is the envelope check of Sec. 7.5.3 applied to a model that cannot tell you its own envelope, so you compute it from the training set instead.

Prefer a model that reports its own uncertainty. Some do, and the difference in what you can build is large: an output that comes with a range lets the service degrade rather than guess. Probabilistic approaches to health diagnostics and prognostics are reviewed with exactly this framing [14], and the hybrid family of Chapter 4 exists partly for this reason – a physics core that stays inspectable with a learned correction on top keeps half the argument available [15].

Regulated domains have gone furthest in codifying what evidence a learned model owes, because they had to. In healthcare, credibility assessment is identified as the most challenging part of getting an in-silico method accepted, and the existing standard's assumption that a one-off assessment suffices is noted as workable for knowledge-driven models and unresolved for data-driven ones [2]. Whatever your domain, that is the honest state of the art, and Chapter 8 takes it further.

7.5.6 What cannot be validated, and saying so

Four gaps that no amount of budget closes. Naming them in the credibility argument is what distinguishes a document a reviewer trusts from one they start probing.

Rare events. You cannot validate a prediction about a failure that has never happened. The demonstrator has never had a pump seize; the twin's behaviour when it does is a claim, not a finding. Validation can only cover the regime your data covers, and the honest sentence is "the fault detector has been tested against manually induced dose failures, not against a real pump failure."

Extrapolation. Every clause of Sec. 7.5.3's envelope was derived from data inside the envelope. The A4 row admits it: three weeks tested, four weeks claimed. One week of extrapolation is a judgement, not a result, and it should be labelled as one.

Counterfactuals. A what-if service predicts what *would* have happened under a schedule that was never run. There is no measurement to compare it with, ever. The only validation available is indirect: show that the model predicts well on schedules that *were* run, and argue that the counterfactual is inside the same envelope. That argument is weaker than it looks and should be stated as what it is.

A model that changes faster than you can validate it. Chapter 2 Sec. 2.13 raised the weekly-retrained model and noted it does not change the twin's classification but changes this chapter's problem substantially: a model that changes weekly must be re-validated weekly. If your validation procedure takes three weeks and your model updates every week, you do not have a validation procedure – you have a backlog. This is the central scaling problem of twins built one at a time, and it is recognised as such: moving from artisanal to industrial twin production puts new demands specifically on the speed and robustness of validation, verification and uncertainty quantification [16]. Chapter 14 owns the operational answer; the design-time answer is to size your validation procedure to your update cadence, or slow the cadence.

7.5.7 What to ask

- What data was this validated on, and had the model seen any of it?
- What accuracy target was it held to, and where did that number come from?
- Which envelope clauses are tested and which are admitted?
- What observation would show this model is wrong?
- If the model is learned: what did the training data cover, and what happens outside it?

7.6 Uncertainty: how wrong might it be

Validation produces a verdict. Uncertainty quantification produces a *number attached to every answer*, and it is what turns a prediction into something a decision can be made against.

7.6.1 Four sources, in plain words

The field splits uncertainty into two families and calls them **aleatory** (the world's own randomness – it will not go away if you learn more) and **epistemic** (your ignorance – it will) [1]. You will meet the two words; they are worth recognising and not worth dwelling on. Four practical sources matter more:

Source	In the demonstrator	Reduced by	Measured in
Measurement noise	The moisture reading wobbles a few units	A better sensor; averaging	Sec. 7.4.1's spreads
Parameter uncertainty	`k_day` is 4.28 plus or minus 0.21	More calibration data, by Chapter 6's `1/sqrt(N)` law	Sec. 7.4.6

Source	In the demonstrator	Reduced by	Measured in
Model-form error	The pot does not actually dry at a constant rate	A better model; nothing else	Sec. 7.4.6's residual spread
Input and scenario uncertainty	You do not know next week's weather, or whether somebody will water by hand	Nothing. It is the world	Not measurable from past data

Model-form error is the one to watch, because it is the only one that cannot be reduced by collecting more of the same data, and it is the one that hides. Fitting more nights improves your knowledge of `k` and does nothing about the fact that drying is not linear. The literature is candid that separating parameter uncertainty from model-form uncertainty is difficult in general [13], which is a good reason not to promise a clean decomposition to a sponsor.

7.6.2 What Monte Carlo can and cannot do

Chapter 6 Sec. 6.7 gave you the mechanism: sample the uncertain inputs many times, run the model on each sample, treat the spread of outcomes as the answer. Chapter 6 Sec. 6.7.4 then said, honestly, that Monte Carlo propagates *stated* uncertainty and cannot discover unstated uncertainty, and handed the question of where the stated ranges come from to this chapter.

They come from Sec. 7.4 and Sec. 7.5. That is the whole answer, and it is worth stating as a chain, because it is the chain a reviewer will walk:

```
calibration campaign  ->  parameter value + spread   (Sec. 7.4.1)
hold-out test         ->  model-reality gap          (Sec. 7.4.6)
envelope work         ->  the conditions all of the above applies in (Sec. 7.5.3)
                          |
                          v
              Monte Carlo propagates these
                          |
                          v
              a prediction with a range on it
```

And so the failure mode is now obvious. A team that runs ten thousand Monte Carlo samples over parameter ranges that somebody guessed has produced a beautifully converged answer to a question nobody asked. The compute is real, the convergence is real, and the range is fiction. Chapter 6 gave you the cost law for those ten thousand runs; this chapter gives you the reason to check the inputs before paying it.

7.6.3 Reporting a prediction, and the double-counting trap

The twin predicts the evening reading. Starting from a morning reading of 640, the three-parameter model says 640 - 33.4 = **606.6**.

Now attach a range. The tempting move is to list every source from Sec. 7.6.1 and add them up:

- parameter uncertainty in `k_day`: the uncertainty in the average rate is 0.21 / square root of 5 = 0.09 per hour, so 0.09 x 7.8 = **0.7** units over the window;
- numerical error at `h` = 0.375: **0.3** units;
- measurement noise on the morning reading: **1.5** units;
- model-reality gap from week 3: **2.0** units.

Total, adding in the usual way for independent contributions – square each, add, take the square root – gives the square root of (0.49 + 0.09 + 2.25 + 4.0) = square root of 6.83 = **2.6** units.

That number is wrong, and the reason is the most useful thing in this section. The week 3 residual spread of 2.0 was measured *end to end*: it was computed from real predictions made with those parameters, at that step size, against readings from that sensor. It **already contains** the measurement noise, the numerical error and the parameter uncertainty. Adding them again counts them twice.

> **The rule.** An end-to-end measurement of the gap supersedes a component-by-component budget of the same gap. Build the budget only for the sources the end-to-end measurement could not have seen.

So the honest report is:

```
Predicted evening reading: 606.6, plus or minus 2.0 (one spread),
about plus or minus 4 for a range that will hold on most days.
```

What the component budget is still for. Two things. First, apportionment: knowing that parameter uncertainty contributes only 0.7 of the 2.0 tells you that another six nights of calibration would barely move the total, and that the money should go on a better model instead. That is a budgeting decision the end-to-end number alone cannot support. Second, and more importantly, the budget lists what is **not** inside the 2.0: anything week 3 did not contain. A repotted plant, a heatwave, a reading near saturation, a hand-watering nobody logged. Those are envelope questions, not range questions, and no arithmetic converts one into the other.

That distinction – a range for what you have measured, an envelope for what you have not – is the honest core of uncertainty quantification, and it is why the national-academies review of digital twins treats VVUQ as a foundational research need rather than a solved procedure, recommending it be made an integral part of new twin programmes rather than a stage at the end [17].

7.7 The credibility argument: what you actually deliver

Everything so far produces evidence. This section produces the artifact, and the artifact is the deliverable – Chapter 2 said credibility evidence is optional for a shadow and mandatory for a twin, and this is the shape the mandatory thing takes.

The word for the bundle is **credibility assessment**: verification and validation establish that the twin meets its intended purpose, and uncertainty quantification supplies a measure

of performance that users apply as part of that assessment, applied across the twin's whole life cycle rather than once at the end [5].

7.7.1 Five parts, and a document that fits on two pages

Credibility argument. A written statement of what the twin claims, for what use, on what evidence, with what limits, and when the claim expires.

1. **The claim.** One sentence per output, in the units of the decision. Not "the model is accurate to 2 units" but "the fault detector catches watering shortfalls of 16 per cent or worse."
2. **The intended use.** Which decision, made by whom, with what authority. Advisory or closed-loop. This is the row of Sec. 7.1.2's table you are claiming, and it sets how much of the rest is required.
3. **The evidence.** Verification performed, calibration campaigns with their data windows, hold-out results with bias and spread, the convergence table and the step size used. Each with a pointer to the run that produced it, which is why Sec. 7.3.4 was a precondition.
4. **The limits.** The validity envelope, clause by clause, marked tested or admitted. The known blind spots – the 10 per cent shortfall, the displaced dripper. What has never been observed at all.
5. **The expiry.** The condition under which this argument stops holding: a date, a parameter refit, a change to the physical twin, a model version bump, or a monitor firing.

Two pages. If it runs to twenty, the twin is doing too many things and should have one argument per output.

7.7.2 Expiry is the part that makes it a twin document

Figure 7.3 is the difference between the two documents, and it is the reason this chapter ends with a document rather than a number.

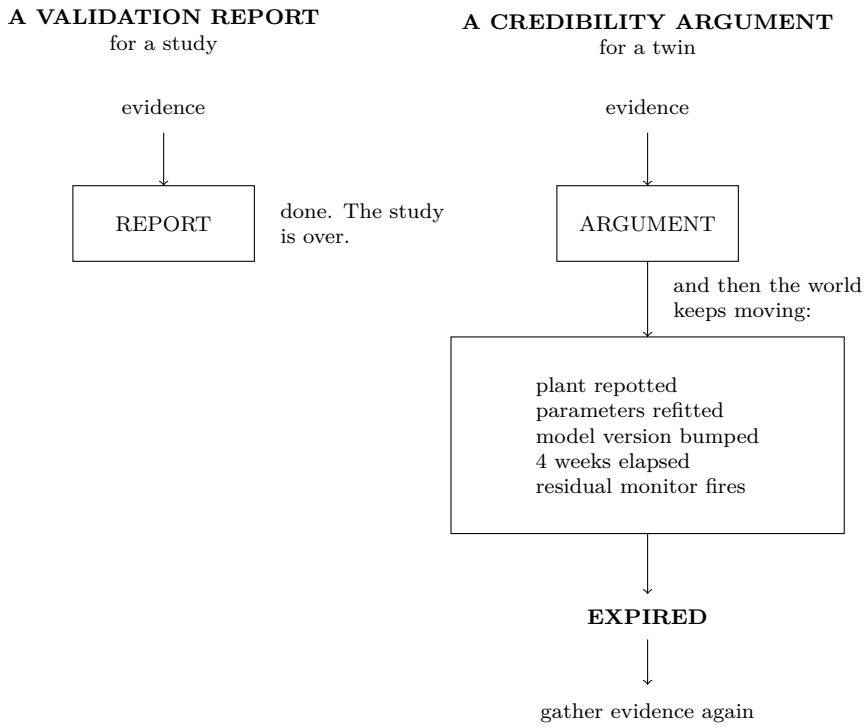

Figure 7.3 Why a twin's trust document has a loop in it. The claim is about a system that is still changing, so the argument has to name the conditions that end it.

A validation report for a simulation study is finished when it is written, because the study is over. A twin keeps running against a system that changes, so an argument with no expiry condition is a claim about a moment that has passed. This is the difference Sec. 7.4.7 drew from [3], stated as a document requirement.

Make the expiry a *condition*, not only a date, and give each one an owner:

Expiry trigger	Detected by	Owner
4 weeks since last calibration (A4)	Calendar	Whoever owns the model registry
Physical twin changed: repotted, plant replaced, dripper moved	A human telling you, which means a process	Lab operations
Model version changed	Release pipeline; golden trajectories diff	You
Residual bias monitor fires (Sec. 7.7.4)	The twin itself	On-call
Sensor replaced or recalibrated	Asset records	Whoever owns the hardware

The second row is the one that fails in practice. There is no signal in the data that says "somebody repotted the plant" – there is only a step change in the residuals a fortnight later, by which time the twin has been quietly wrong. Chapter 3's identity and binding concern is where the mechanism belongs; here it is enough to notice that a credibility argument with no process behind row 2 is resting on somebody remembering.

7.7.3 The record the argument depends on

Chapter 3 Sec. 3.2.2 posed two questions that look similar and are not: "what was measured at time t?" and "what did the twin *believe* at time t?" Chapter 3 Sec. 3.2.7 said every command must be audited with the state that justified it – the twin state, its age, the model version, and the service that asked – and warned that the record is unrecoverable if you did not write it at the time.

Here is why the credibility argument needs it. Six months in, an alert is disputed. The questions asked, in order, are:

1. What did the twin predict, and what did it measure? (Both, separately. Chapter 5 Sec. 5.2.4 already insisted that predicted and measured live in different columns.)
2. Which model version and which parameter set produced that prediction?
3. When were those parameters last fitted, on what data window?
4. Was the input inside the validity envelope at the time?
5. Was the credibility argument current, or had it expired?

Every one is answerable from a store designed for it and none is reconstructable later. **The credibility argument is a claim about a system's behaviour; the audit record is what makes the claim checkable; and Chapter 3 is where you had to build it.** Chapter 10 does the data engineering properly.

7.7.4 Monitors that keep the argument alive

The argument is written once and then has to survive contact with production. Three monitors are cheap, and each maps to something this chapter established.

Residual bias. You are already computing residuals for the fault detector. Track their rolling average. Week 3 established that the bias should be 0.0 with a spread of 2.0. A rolling 14-day average residual outside plus or minus 1.0 is a refit trigger – the average of fourteen values with spread 2.0 should sit within about 2.0 / square root of 14 = 0.53 of zero, so 1.0 is comfortably outside ordinary variation and comfortably inside the point where the fault detector suffers.

Residual spread. A growing spread with no bias is a different disease – usually a sensor degrading or the plumbing dropping samples. Same data, one more line of code, and it distinguishes two causes that look identical on a dashboard showing "error going up".

The estimator gain. Chapter 6 Sec. 6.6.2 defined the gain as how far the state estimator pulls toward the measurement, said that Chapter 3's monitoring should carry it, and said in as many words that Chapter 7 would ask about it. Asking:

> **What has the gain been doing?** A gain drifting toward 1 means the estimator is trusting the sensor and abandoning the model, which is the model-reality gap widening under a different name. A gain drifting toward 0 means the twin has stopped listening to reality, which is either a correct response to a broken sensor or a twin that has quietly become a simulation. **Either drift invalidates the credibility argument, because the argument was written about a system with a different balance between model and measurement.** Put it on the same dashboard as the residuals and treat a sustained move as an expiry trigger.

7.7.5 The contract clause

Chapter 2 Sec. 2.11 asked you to write a contract clause that could be checked from a deployment diagram, and its solution conceded the cost of that choice: the test says nothing about whether the twin is any good, which is why the contract also needs this chapter's clauses. Here they are, in the form of what to require rather than legal wording:

- **A credibility argument in the five-part form, delivered with the software**, naming the specific outputs it covers. A vendor who supplies accuracy figures without an intended-use statement has supplied a marketing number.
- **The validation data, or the right to run your own validation.** If neither is available, the twin's claims are unverifiable by you, and that should be priced.
- **The envelope, machine-readable.** Ranges per input, so your service can enforce them (Sec. 7.5.3). A prose envelope is not enforceable in code.
- **A stated expiry condition and a re-validation obligation**, including what happens to the argument when the vendor ships a model update.
- **Notification of model changes**, with golden-trajectory diffs. A silent model update is a silent change to every claim in the argument.

The industry conversation is moving toward making this machine-readable at the ecosystem level, since twins increasingly need to trust each other rather than only their owners: one proposal is a "trust vector" in which each side publishes, per trustworthiness characteristic, a link to a human- and machine-readable justification – an assurance case, for example – so that a composed system can reason about whether its parts are trustworthy enough for the job [18]. Chapter 15 is where twins start composing; the clause above is the single-twin version of the same idea.

7.7.6 What this document is not: the safety case

If the twin is closed-loop – Chapter 2 Sec. 2.8's world, and Chapter 1's observation that the top rung needs "everything in Chapter 7 and a safety argument" – then a second document exists, with a different owner, a different standard and a different question.

	Credibility argument	Safety case
Question	Is the twin's answer good enough for this decision?	Is the system acceptably safe, including when the twin is wrong?
Assumes the twin is right	Yes, within stated limits	**No.** It assumes the twin fails and asks what happens
Owner	The twin's engineering team	The system safety function
Chapter 3 component it leans on	The store and the model registry	The actuation guard and fail-safe behaviour

The credibility argument is an input to the safety case, not a substitute for it, and the most important thing it contributes is Sec. 7.5.6's list of what cannot be validated – because that list is exactly the set of conditions the safety case has to handle without the twin's help. A safety engineer handed a credibility argument with no limitations section will either reject it or, worse, believe it.

Do not write the safety case as a side effect of writing this one. Say which document you are producing.

7.8 Worked example: the credibility argument for the greenhouse twin

Everything above, assembled, for the demonstrator at Chapter 2's Stage 2 – the fault detector, advisory, human in the loop.

7.8.1 Fix the intended use first

From Sec. 7.1.2's table this is **Diagnose**, advisory. Evidence owed: verification, calibration, a hold-out test, a stated threshold with its miss rate. Not owed: fail-safe analysis, runtime envelope enforcement on a command path, a safety case. Those arrive at Stage 3 and it is worth noticing how much cheaper Stage 2 is – one of Chapter 1's reasons for climbing the ladder one rung at a time.

7.8.2 Verification, in an afternoon

- Read the nine lines. Units confirmed: `g` per 100 ml, and the code divides the dose by 100. Signs confirmed. Dose applied at the start of the containing step, documented as a known 45-minute placement error at `h` = 1, which disappears at the step size chosen below.
- Convergence run on the 24-hour dry-down (Sec. 7.3.2's table). Chose `h` = 0.375 hours, numerical error 0.3 units.
- Plumbing: fed a constant 600 through the connector, store and runner; confirmed 600 arrives at the model input with the right timestamp and no forward-fill across an induced 90-minute gap. **This test failed the first time**, which is the ordinary outcome and the reason the test exists.
- Recorded four golden trajectories: a dry-down, a normal day, a missed evening dose, a saturation excursion.

7.8.3 Calibration

Two campaigns, deliberately separate (Sec. 7.4.4):

Parameter	Experiment	Value	Spread	Fitted on
`k_night`	6 dry overnight windows, 12 h each	3.67 units/h	0.26	Week 1
`k_day`	5 daytime windows, 7.8 h each	4.28 units/h	0.21	Week 2
`g`	5 doses of 100 ml, readings 10 min either side	47.0 units/100 ml	2.55	Week 1

Identifiability check: each parameter has an experiment in which the others do not act. A single fit on daily net change was attempted and abandoned – see Sec. 7.4.4 for the three parameter pairs it could not separate.

7.8.4 Validation

Week 3, untouched by either fit. Day-ahead prediction over the 7.8-hour window:

```
bias   = 0.0 reading units
spread = 2.0 reading units
```

Threshold derivation (Sec. 7.5.2): expected step for a 120 ml dose is 56.4; spread of a healthy step is 3.0; threshold at three spreads is 9.0; alert when the observed step is below 47.4. **Smallest detectable shortfall: 16 per cent.**

7.8.5 The two-page argument

> **Claim.** For the plant-pot rig in bay 3, the twin's watering-fault detector raises an alert when a scheduled dose delivers 16 per cent less water than commanded, or less. Its day-ahead moisture prediction is accurate to plus or minus 2.0 reading units, with no detectable bias.
>
> **Intended use.** Advisory. The alert notifies the duty researcher, who inspects the rig. The twin issues no commands. Chapter 1's value metric: experiment-weeks lost to undetected watering faults.
>
> **Evidence.** Verification: code review, convergence table (`h` = 0.375 h, numerical error 0.3 units), end-to-end plumbing test with induced gap, four golden trajectories under version control. Calibration: two campaigns, week 1 and week 2, parameters and spreads as tabulated, held in the model registry as parameter set `pot-3/v2`. Validation: week 3 hold-out, bias 0.0, spread 2.0, five day-ahead predictions. Threshold derived from the value metric, not chosen.
>
> **Limits.** Valid for readings between 350 and 900 (sensor datasheet, untested by us, enforced at the service boundary). Valid for prediction horizons up to 12 hours; 24-hour horizons untested. Cannot distinguish a failed pump from a displaced dripper (assumption A3); both are treated as "water did not arrive", which the value metric tolerates. Does not detect shortfalls below 16 per cent; these are not expected to affect plant health. Makes no claim about surface pooling or layered drying (assumption A5). Never observed: a real pump failure, a heatwave, a reading above 900.
>
> **Expiry.** Whichever comes first: 4 weeks from 2026-08-09; any change to the rig, plant or sensor; a model or parameter-set version change; the 14-day rolling residual bias leaving plus or minus 1.0 reading units; the estimator gain moving outside 0.4 to 0.85 for more than 48 hours.

7.8.6 What this cost, and the point of the section

Two weeks of logging that was happening anyway, one week held back, an afternoon of verification, a morning of arithmetic, and two pages. **No Bayesian machinery, no sensitivity analysis, no Monte Carlo.** Chapter 6 Sec. 6.9 closed by noting that it

chose the cheaper option in five of six subsections, and said that was the chapter's thesis in miniature. The same applies here: a credibility argument for an advisory twin is a fortnight of discipline, not a research programme. Sec. 7.1.2's table is what keeps it that way, and the twin that needs more is the twin that has been allowed to do more.

7.9 Faded example: the offshore turbine, trusted

Chapters 4, 5 and 6 each took the offshore turbine further. Chapter 4 chose its model families, Chapter 5 sized its runs, Chapter 6 chose its simulators. Now decide whether anyone should believe it. Two parts are worked; four are yours.

The system, recapped. A monitoring twin for a floating offshore wind turbine. A reduced-order structural model with a small number of degrees of freedom, a Kalman filter estimating loads from a handful of sensors, and a fatigue-accumulation service that reports remaining life. Decisions: when to send a maintenance vessel, and whether to extend the certified life. Real systems of this shape exist and are validated against full-scale prototype measurements [19].

(a) Which row of Sec. 7.1.2 – worked. Two outputs, two rows, and this is the answer that matters most.

The maintenance-scheduling output is **Decide (advisory)**: a human dispatches the vessel, a wrong answer costs a wasted sailing or a missed inspection window. The life-extension output is different in kind. It feeds a certification decision with a regulator on the other side, which is Chapter 1's **Certify** pattern, and the evidence owed steps up sharply – traceable provenance for every input, an argument a third party can audit, and justified uncertainty on the fatigue number rather than a point estimate. **Two outputs of one twin, two credibility arguments.** Writing one document covering both would have to satisfy the harder row throughout, which is the expensive mistake Sec. 7.1.2 warned about, in the other direction.

(b) Where the validation data comes from – worked, because it is the hard part. You cannot hold out a week of turbine failures; there are none. Three sources, in decreasing strength:

1. **A full-scale instrumented prototype.** Compare twin outputs against measurements from the real structure. This is the strongest evidence available and it is what the published TetraSpar work does, comparing digital-twin outputs against tower-bottom moment and wind measurements from the prototype [19]. Note the honesty in that comparison: the measured wind speed is read at a nacelle anemometer, which is *expected* to disagree with the rotor-averaged figure the twin estimates, so the two quantities are not the same quantity and the report says so. **A validation comparison between two things that are not the same quantity is a trap, and naming the mismatch is the fix.**
2. **A higher-fidelity model as a stand-in.** The reduced-order model can be validated against the full finite-element model it was reduced from. This validates the *reduction*, not the physics – Chapter 6 Sec. 6.5.1 made the same point about surrogates – and the credibility argument must not let the two blur.

3. **Sister assets.** Other turbines in the same farm, which is a real technique and a real complication, since the population is not homogeneous.

Now yours.

(c) The fatigue service integrates load estimates over years. Sec. 7.6.3 warned about double counting; this output has the opposite problem. Explain why an end-to-end hold-out is unavailable here in a way it was available for the pot, and what evidence has to replace it. *Hint: how long is the hold-out period for a claim about a 25-year life, and what does that force you to validate instead of the final number?*

(d) Chapter 6 gave this turbine a Kalman filter. Write the gain monitor for it, following Sec. 7.7.4: what would a gain drifting toward 1 mean for a structural load estimate specifically, and why is it more alarming here than in the pot?

(e) The reduced-order model is a surrogate. Using Sec. 7.5.5's table, say what its envelope has to be derived from, and name one input dimension along which "outside the training range" is likely and detectable at the service boundary.

(f) Write the expiry conditions. At least four, and at least one that no data stream can detect. *Hint: what happens to a floating structure's dynamics after a mooring line is replaced?*

7.10 Posed problem: the V&V plan for a pumping station

No solution is given. This is the deliverable this chapter exists to make you capable of producing, and it is deliberately posed at the point in a project where the information is incomplete.

The situation. A municipal water utility operates a pumping station: four pumps, a wet well with a level sensor, inflow that varies with rainfall, and a control system that starts and stops pumps on level thresholds. A vendor proposes a twin that will (i) detect a failing pump from its power draw and flow, (ii) predict wet-well level six hours ahead from a rainfall forecast, and (iii) in a later phase, set the pump start/stop thresholds automatically to reduce energy cost.

The plant's Supervisory Control and Data Acquisition (SCADA) system holds the only long record of how it has run. The vendor's proposal says the model "has been validated against six months of historical SCADA data and achieves 94 per cent accuracy".

Produce a V&V plan of no more than four pages containing:

1. **A row of Sec. 7.1.2's table for each of the three outputs**, with the evidence each owes. Note explicitly where output (iii) changes the answer for outputs (i) and (ii), and why.
2. **Three questions about the vendor's sentence** that must be answered before it means anything. At least one about the data, one about the metric, and one about the ordering trap of Sec. 7.2.2.
3. **A calibration and validation data plan**: what data, over what period, held out how, and what conditions the hold-out must contain that the calibration set does not. Rainfall is your friend here.

4. **An identifiability analysis** for at least one parameter pair you suspect the historical data cannot separate. *Prompt: the twin must infer both pump efficiency and pipe friction from power and flow. When do those two effects move together?*
5. **An acceptance threshold for output (i)**, derived in Sec. 7.5.2's five steps from a value metric you state yourself, including the smallest fault it will miss and an argument that missing it is acceptable.
6. **The validity envelope**, with each clause marked tested or admitted, and the runtime check each tested clause implies.
7. **The expiry conditions**, with owners.
8. **One paragraph on what you will not be able to validate**, and what the later closed-loop phase must therefore handle without the twin's help.

What a good answer looks like. It is specific about data rather than about process. It notices that "94 per cent accuracy" has no units, no decision attached, and no statement of what was held out – and it asks rather than assumes. It notices that a six-hour level prediction and a pump-fault detector are validated against different things on different time scales. It states at least two things it cannot validate. And it costs the plan: the honest answer to output (iii) is that closing the loop on a municipal asset requires a safety argument this plan does not contain, and saying so early is worth more than any amount of the plan.

7.11 Summary

Seven things, tied to the five objectives.

1. **Trust is not a property of a model** (Sec. 7.1). The answerable question names an output, a decision, a set of conditions and a body of evidence. How much evidence you owe is set by what the twin is allowed to do – Sec. 7.1.2's table – and the inference runs in that direction, not the other. *(Objective 1, 5.)*
2. **Four words, one order** (Sec. 7.2). Verify, calibrate, validate, quantify. Calibrating before verifying lets the parameters absorb your bugs; validating on the calibration data measures the fitting procedure; fixing the model with the hold-out data burns the hold-out data. *(Objective 1.)*
3. **Verification is your job and needs no statistics** (Sec. 7.3). Three layers – model, solver settings, plumbing – and the plumbing produces the most defects and gets the least attention. The convergence table is the tool that separates numerical error from the model-reality gap, and until the first is a small fraction of the second, a validation run measures your step size. *(Objective 4.)*
4. **A parameter without a spread is not a calibration result** (Sec. 7.4). Fitting `k` from six nights gave 3.67 plus or minus 0.26. Two parameters need two experiments, because data in which both act together may not be able to tell them apart – and three candidate pairs that fit daily data identically predicted watering steps from 46.8 to 68.4. *(Objective 2.)*
5. **Residuals have two numbers and they mean different things** (Sec. 7.4.5). Bias is a model wrong in a repeatable direction and is good news. Spread is noise. A bias of -4.8 against a spread of 1.67 named its own cause in the assumption ledger; the extra parameter that fixed it was earned because it removed the bias on data neither fit had seen. *(Objective 3.)*

6. **The accuracy target comes from the decision** (Sec. 7.5.2). Five steps from a value metric to a threshold to the smallest fault that threshold can catch – 16 per cent here – and then the question of whether missing the rest is acceptable. A 1.0-unit calibration error, invisible in any fit-quality number, moved that threshold by 13 per cent. *(Objective 4.)*
7. **The deliverable is a two-page argument with an expiry condition** (Sec. 7.7). Claim, intended use, evidence, limits, expiry. It depends on an audit record you had to build in Chapter 3, it is kept alive by three monitors that cost a few lines each, and it is an input to the safety case rather than a substitute for it. *(Objective 5.)*

And the honest one, which Chapter 8 will hear again in a harder form. Everything in this chapter establishes what a model does in conditions somebody has observed. Nothing in it establishes what it will do in conditions nobody has. The range comes from measurement; the envelope comes from admission; and telling a sponsor which is which is the most valuable thing you will do with this material.

7.12 Exercises

Solutions or hints follow. Each exercise names the objective it exercises.

7.12.1 *(Objective 1.)* Classify each of the following as verification, calibration, validation or uncertainty quantification: (a) "the model predicts a 4 per cent step for a dose that should give 40 per cent – check whether we are dividing by 100 twice"; (b) "run it against last month's data that we did not fit on"; (c) "the answer is 606.6 but I want a range on it"; (d) "find the value of `k` that best matches these six nights"; (e) "the solver gives a different answer at half the step size".

7.12.2 *(Objective 1.)* A colleague says: "we fitted the model on all twelve weeks of data and it matches beautifully – 98 per cent of predictions are within 3 units." Name three things wrong with that sentence as a credibility claim, and write the one question that exposes the most important one.

7.12.3 *(Objective 2.)* Fit `k` from these four dry overnight windows, each 10 hours: drops of 34, 41, 37 and 40 reading units. Report the value and the spread, both per hour. Then state how many nights you would need to halve the uncertainty in the average.

7.12.4 *(Objective 2.)* A twin of a heated tank has two parameters: the heater's power delivered to the water, `p`, and the tank's heat loss rate to the room, `c`. The only data available is the steady temperature the tank settles at with the heater on continuously. Explain why this data cannot identify `p` and `c` separately, and design the smallest additional experiment that can.

7.12.5 *(Objective 3.)* A hold-out test on ten days gives residuals: +2, -1, +3, 0, +2, +1, +3, -1, +2, +4. Compute the bias and the spread. Decide whether this model needs a new parameter or better data, and say what you would look at next.

7.12.6 *(Objective 3.)* You add a parameter to fix a bias, refit, and the bias on the same data drops to zero. Your colleague proposes shipping. Write the two-sentence objection, and say what evidence would change your mind.

7.12.7 *(Objective 4.)* A dose of 150 ml is expected to step the reading by `g` x 1.5 = 70.5, and healthy steps for that dose have a spread of 3.8. Set a three-spread threshold, compute the smallest percentage shortfall it detects, and state one condition under which you would use two spreads instead.

7.12.8 *(Objective 4.)* Your convergence table shows numerical errors of 8.0, 4.1, 2.0 and 1.0 units at successively halved step sizes. Your hold-out spread is 1.5 units. Which step size may you validate at, and what does that do to the run time relative to the coarsest setting? Then explain what you would suspect if the errors had been 8.0, 7.9, 7.9 and 7.8.

7.12.9 *(Objective 5.)* Write the five-part credibility argument, at one paragraph per part, for a twin that predicts a building's next-hour electricity demand to inform a human operator's decision to pre-cool. Invent the numbers, but make them internally consistent, and make at least two of your envelope clauses admitted rather than tested.

7.12.10 *(Objectives 1-5, and the one that leaves the book.)* Take the real plant-controller rig. Log two weeks. Fit `k` and `g` using Sec. 7.4.1's and Sec. 7.4.2's two campaigns on your own data. Hold out a third week. Report bias and spread, and compare them with this chapter's constructed figures. Then write down the three things about the real data that this chapter's tables did not prepare you for.

Solutions and hints

7.12.1. (a) Verification – comparing code against intent, no measurement of the world involved. (b) Validation. (c) Uncertainty quantification. (d) Calibration. (e) Verification – specifically the convergence check of Sec. 7.3.2, and note that this one is often *misreported* as the model being inaccurate, which is the confusion the section exists to prevent.

7.12.2. Three faults: (i) no hold-out – the model was fitted on all twelve weeks, so 98 per cent measures the fit, not the model; (ii) "within 3 units" is in the model's units, not the decision's, so nobody can tell whether it is good enough for anything; (iii) no envelope and no expiry – twelve weeks of what conditions, and for how long does the claim hold? The question that exposes the most: **"which data did you hold back?"** If the answer is none, the number means nothing at all and the other two faults are academic.

7.12.3. Average drop = (34 + 41 + 37 + 40) / 4 = 152 / 4 = 38.0, so `k` = 38.0 / 10 = **3.80 units per hour**. Deviations from 38: -4, +3, -1, +2; squares 16, 9, 1, 4, sum 30; divide by 3 gives 10; square root gives 3.16 per night, so the spread of `k` is 3.16 / 10 = **0.32 units per hour**. Uncertainty in the average = 0.32 / square root of 4 = 0.16. To halve it to 0.08 you need four times as many nights: **16 nights**, by Chapter 6's `1/sqrt(N)` law. Note how expensive that is – more than two weeks of suitable nights – which is usually the moment a team decides 0.16 is fine.

7.12.4. At the steady temperature nothing is changing, so the heater's input exactly balances the loss to the room: the data constrains only the *ratio* of `p` to `c`. Doubling both gives the identical steady temperature, so infinitely many pairs fit – Sec. 7.4.4's situation exactly. The smallest additional experiment: **turn the heater off and record how fast the tank cools.** Cooling with no heater involves `c` alone, which pins it; the steady temperature then pins `p`. The general principle is the one that separated `k` from `g`: find a condition in which one parameter acts and the other does not.

7.12.5. Sum = 2 - 1 + 3 + 0 + 2 + 1 + 3 - 1 + 2 + 4 = 15, so bias = **+1.5**. Deviations

from 1.5: 0.5, -2.5, 1.5, -1.5, 0.5, -0.5, 1.5, -2.5, 0.5, 2.5; squares 0.25, 6.25, 2.25, 2.25, 0.25, 0.25, 2.25, 6.25, 0.25, 6.25, sum 26.5; divide by 9 gives 2.94; square root gives spread **1.72**. The bias is about 0.9 spreads – present but not dominant, unlike Sec. 7.4.5's -4.8 against 1.67, which was nearly three. **Neither answer is forced by the arithmetic**, and that is the point of the exercise: with a bias smaller than the spread, look first for a *named* cause in the assumption ledger. If one exists (a systematic effect you can point at), the parameter may be earned. If none does, adding a parameter is fitting noise, and the honest next step is more hold-out days to see whether the bias persists.

7.12.6. Objection: "the bias dropped on the data you used to fit the new parameter, which is guaranteed by arithmetic and is not evidence; and the data that revealed the bias is now calibration data, so we currently have no hold-out at all." What would change my mind: the same bias staying at zero on a fresh period neither the original fit nor the fix has seen, plus a named row of the assumption ledger that the new parameter corresponds to.

7.12.7. Three spreads = 3 x 3.8 = **11.4** reading units; alert when the observed step is below 70.5 - 11.4 = **59.1**. Smallest detectable shortfall = 11.4 / 70.5 = **16 per cent** – the same figure as Sec. 7.5.2, which is not a coincidence, since both the expected step and its spread scale with dose size. Use two spreads when a missed fault costs far more than a false alarm: a plant you cannot replace, or an unattended rig over a holiday. The price is more false alarms, and the thing to state is the new miss floor (2 x 3.8 / 70.5 = 11 per cent) so the trade is visible.

7.12.8. Errors halve as the step halves, so the rule is first order and behaving. To get the numerical error to a fifth of the 1.5-unit hold-out spread you need it below 0.3, so the 1.0 row is not enough: **one more halving to about 0.5, and another to about 0.25**, i.e. two steps beyond the finest row shown. Relative to the coarsest setting that is 32 times the steps, and Chapter 5 Sec. 5.3.4's arithmetic tells you whether that still fits the request budget. If instead the errors had barely moved – 8.0, 7.9, 7.9, 7.8 – the run is **not converging**, and step size is not the cause of the 8.0. Suspect an event landing on a step boundary, a discontinuity the solver is stepping over, or a defect in the model code; Sec. 7.3.2 says stop and get help rather than halving again.

7.12.9. *Hint, not a solution.* Check yours against four tests. (i) Is the claim in the units of the decision – kilowatts, or "enough warning to pre-cool by 2 degrees" – rather than in a percentage? (ii) Does the evidence paragraph name a hold-out period and the bias and spread on it? (iii) Do your admitted limitations include at least one condition you have never observed – an unusually hot day, a bank holiday occupancy pattern? (iv) Does the expiry paragraph contain a condition that no data stream can detect, such as a change of tenant or a new chiller? If all four hold, the document would survive a reviewer.

7.12.10. No solution – that is the point. Two predictions about what you will find. First, your data will have gaps, and the gaps will be where something interesting happened, because the same event that disturbed the plant disturbed the logger. Second, at least one of your six "dry nights" will turn out not to have been dry, and finding that is worth more than the parameter it changes.

7.13 Where to go next

In this book. Chapter 8 closes Part II by taking the data-driven family and its specific risks seriously, including what Sec. 7.5.5 could only sketch about validating a model whose parameters mean nothing. Then Part III begins the build. Chapter 9 is the connector that supplies the measurements every test here needed. Chapter 10 is the store and the provenance that Sec. 7.7.3 depends on and does not build. Chapter 11 turns Sec. 7.5.2's threshold and Sec. 7.7.4's monitors into services. Chapter 12 asks what a platform gives you of all this, and Chapter 13 covers the standards – ASME VV-40 and ISO 23247 among them – that this chapter named once each and did not explain. **Chapter 14 is the one this chapter leans on hardest**: expiry conditions are only worth writing if something acts on them, and the operational loop that refits, re-validates and re-issues the credibility argument is Chapter 14's subject. Chapter 15 asks what happens to a credibility argument when twins start composing, which is where [18]'s trust vectors point.

In the literature, if you want more. These are the sources drawn on above, grouped by what you would read them for.

- *Calibration, properly:* [4] is the closest thing to a textbook treatment aimed at twin builders, and goes from least squares through nonlinear methods to design-space exploration. [20] shows the same procedure carried out end to end on a physical incubator, including the comparison of a two-parameter against a four-parameter model – which is Sec. 7.4.6's question with real data. [12] is the matching tutorial, and shows calibration wired in as a *service* inside a running twin.
- *V&V for twins specifically, and how it differs from classical practice:* [3] is the one to read first; it is the source of this chapter's continual-validation framing and surveys the level-by-level approaches. [5] is a short standards gap analysis that names credibility assessment as the hole in ISO 23247. [21] places V&V within the engineering-design lifecycle.
- *Uncertainty:* [13] is the survey, and is candid about what cannot be cleanly separated; [17] is the national-academies review that treats VVUQ as an open research agenda rather than a procedure. [1] is the clearest short statement of what bounds a twin's domain of validity.
- *Testing, from a software engineer's angle:* [6] on automated and systematic twin testing, [7] on multi-level testing frameworks, [8] on carrying a requirement from simulation to reality, and [9] on why continuous integration for cyber-physical systems is harder than you expect.
- *Validation against a real asset:* [19] is the offshore example of Sec. 7.9 and is worth reading for its honesty about what its comparison does and does not show. [10] is the structural-engineering counterpart, built around dedicated measurement campaigns. [22] surveys the uncertainty sources in that domain.
- *Consulted, not drawn on above:* [23] on a principled probabilistic foundation for calibrating and updating twins at fleet scale, [24] and [25] on Bayesian model updating for structures, [26] on a clinical twin that reports its predictions with uncertainty by construction, [27] on the regulatory framing in healthcare, [28] on stating twin fidelity as an equivalence property rather than an error figure, and [16] on what breaks when twins are produced industrially rather than one at a time.

In the demonstrator. Exercise 7.12.10 is the assignment, and it is the one that will teach you the most in this book so far, because it is the first time the numbers will disagree

with you. Run the two campaigns, hold out a week, and write the two-page argument of Sec. 7.8.5 with your own figures in it. Then do the thing that makes it real: put the expiry conditions in a monitor, and wait for one to fire.

7.14 References

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

[2] 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.

[3] 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.

[4] C. Gomes, H. Feng, Z. Kazemi, and K. Pierce, "Calibration of Models for Digital Twins," in *The Engineering of Digital Twins*, J. Fitzgerald, C. Gomes, and P. G. Larsen, Eds., Cham: Springer International Publishing, 2024, pp. 129–146. doi: 10.1007/978-3-031-66719-0_6.

[5] 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.

[6] Y. Ma *et al.*, "Automated and Systematic Digital Twins Testing for Industrial Processes." arXiv, Feb. 2023. doi: 10.48550/arXiv.2302.13198.

[7] I. Aldalur, A. Arrieta, A. Agirre, G. Sagardui, and M. Arratibel, "A microservice-based framework for multi-level testing of cyber-physical systems," *Software Quality Journal*, vol. 32, no. 1, pp. 193–223, Mar. 2024, doi: 10.1007/s11219-023-09639-z.

[8] A. Agrawal, P. Zech, and M. Vierhauser, "Coupled Requirements-Driven Testing of CPS: From Simulation to Reality," in *2024 IEEE 32nd International Requirements Engineering Conference (RE)*, Jun. 2024, pp. 337–344. doi: 10.1109/RE59067.2024.00040.

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

[10] N. Wagner and D. Tcherniak, *Data-Driven Updating of Digital Twins through Experimental Measurements and Parametric Finite Element Model Optimization.* 2025.

[11] G. Abbiati, C. Gomes, M. Sandberg, Z. Kazemi, S. T. Hansen, and P. G. Larsen, "Modelling for Digital Twins," in *The Engineering of Digital Twins*, J. Fitzgerald, C. Gomes, and P. G. Larsen, Eds., Cham: Springer International Publishing, 2024, pp. 89–127. doi: 10.1007/978-3-031-66719-0_5.

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

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

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

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

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

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

[18] "Assuring Trustworthiness in Dynamic Systems Using Digital Twins and Trust Vectors." Digital Twin Consortium, May 2024. Available: https://www.digitaltwinconsortium.org/assuring-trustworthiness-in-dynamic-systems-using-digital-twins-and-trust-vectors-form/

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

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

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

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

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

[24] M. Torzoni, M. Tezzele, S. Mariani, A. Manzoni, and K. E. Willcox, "A digital twin framework for civil engineering structures," *Computer Methods in Applied Mechanics and Engineering*, vol. 418, p. 116584, Jan. 2024, doi: 10.1016/j.cma.2023.116584.

[25] M. Torzoni, A. Manzoni, and S. Mariani, "A Deep Neural Network, Multi-fidelity Surrogate Model Approach for Bayesian Model Updating in SHM," in *European Workshop on Structural Health Monitoring*, P. Rizzo and A. Milazzo, Eds., Cham: Springer International Publishing, 2023, pp. 1076–1086. doi: 10.1007/978-3-031-07258-1_108.

[26] A. Chaudhuri *et al.*, "Predictive digital twin for optimizing patient-specific radiotherapy regimens under uncertainty in high-grade gliomas," *Frontiers in Artificial Intelligence*, vol. 6, Oct. 2023, doi: 10.3389/frai.2023.1222612.

[27] 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.

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