# Aigile: A Working Paper on Human-AI Collaborative Software Engineering

**Subtitle:** Why the agile core survives the agentic era, and how to rebuild software methodology on top of it

**Status:** Working paper, version 0.1 (draft for discussion) **Date:** July 2026 **Intended audience:** Software engineering leaders, architects, methodologists, and practitioners working with agentic AI systems **Purpose of this document:** This paper establishes the conceptual foundation of the aigile approach. It is intentionally foundational. In-depth methodology papers, concrete playbooks, tooling guides, and worked examples are expected to build on the definitions and arguments established here.

---

## Abstract

Spec-driven development (SDD) has emerged as the dominant answer to a real problem: coding agents are powerful but unguided, and unguided generation produces unmaintainable results. The dominant SDD implementations, however, repeat a mistake the software industry already made once. They assume that a sufficiently detailed specification, written up front, can replace shared understanding. That assumption defined the waterfall era, and it failed for reasons that have nothing to do with who, or what, executes the specification.

This paper argues that the core insight of agile software development, namely that correctness in software emerges from short feedback loops rather than from complete upfront description, is not weakened by agentic AI. It is amplified by it. At the same time, agile as practiced by human teams relied on a resource that agents do not have: persistent, accumulating, shared memory. Any methodology for the agentic era must therefore solve a problem classic agile never had to solve.

We call the resulting approach **aigile**. Its defining commitments are: humans always own intent; artifacts are split into a durable intent layer, an ephemeral work layer, and a derived description layer; documentation is treated as load-bearing operational infrastructure rather than communication overhead; verification is restructured around review economics rather than review discipline; drift between specification and system is managed like defects, not treated as sin; and validation, the question of whether the right thing was built, remains structurally human at every level of automation.

The paper develops each of these commitments, defines the protocols that operationalize them, describes how the framework scales from a two-person team to a fully agentic delivery organization, and closes with a research agenda.

---

## 1. Introduction: the moment we are in

Software engineering is absorbing its largest change of production economics since the compiler. Code generation, once the scarce and expensive center of the discipline, is becoming abundant and cheap.

Agentic systems can plan, implement, test, and refactor with limited supervision. The constraint has moved. It no longer sits in the typing. It sits in knowing what to build, verifying what was built, and keeping a truthful shared picture of the system alive over time.

The industry's first structured response to this shift is spec-driven development. Tools such as GitHub Spec Kit, AWS Kiro, and Tessl propose that specifications, not code, should become the primary artifact: write the spec, let agents execute it. The instinct behind SDD is correct. Agents need durable, explicit context, because they have none of their own. But the dominant implementations wrap that correct instinct in a process shape the industry has already tested to destruction: a one-directional pipeline from requirements to plan to tasks to code, in which learning flows forward but never backward.

This paper takes a position: the right foundation for agentic software engineering is not a revival of document-first sequential process. It is a deliberate re-derivation of agile principles under new economic and cognitive conditions. Some agile practices transfer directly. Some must be modified. A small number must be inverted. The work of this paper is to say which, and why, with enough precision that concrete methodology can be built on top.

Two design goals run through everything that follows:

1. **Scale-invariance.** The framework must state its own minimum viable form. A methodology that can only be adopted at maximum weight will be adopted at maximum weight and then blamed for it. This happened to SAFe, and it is happening to SDD.
2. **Mode-invariance.** The framework must describe a single coherent system that spans the spectrum from human-AI pair collaboration to fully agentic delivery, rather than two disconnected methodologies. The difference between these modes should be expressible as positions on a dial, not as a fork.

---

# Part I: Diagnosis

## 2. Spec-driven development and the waterfall relapse

### 2.1 What current SDD gets right

Current SDD tooling encodes several genuinely valuable observations:

- Coding agents perform dramatically better with explicit, written context than with ad hoc conversational prompting.
- Some project knowledge is genuinely foundational: architectural constraints, domain invariants, technology decisions. Making this knowledge explicit benefits every participant, human or agent.
- Decomposing features into small, well-described tasks improves both agent output quality and human reviewability.

Aigile keeps all three of these observations. They are not the problem.

### 2.2 What current SDD gets wrong

The problem is the process shape wrapped around them. In practice, the dominant SDD workflow looks like this: a requirements document is generated, then an implementation plan, then a task list, then code, each stage depending on the previous one. Practitioners who have

tested this shape on real projects report a consistent experience: the workflow is slow and heavy, produces large volumes of markdown that nobody meaningfully reviews, and delivers the same defects as lighter approaches while adding ceremony. Independent evaluations have found the full SDD pipeline to be as much as an order of magnitude slower than iterative prompting with review, without a corresponding quality gain.

The deeper critique is structural. SDD in its heavy form repeats the bet the industry made with large-scale outsourcing: that understanding can be replaced by documentation, that if the description is written clearly enough, the party on the other side will execute it faithfully. That bet failed when the executing party was a contractor on another continent. There is no mechanism by which it succeeds when the executing party is a language model. Documents do not carry intent. They carry text. Intent is reconstructed by the reader, and the fidelity of that reconstruction is exactly what short feedback loops exist to check.

There is also a propagation problem specific to generative systems. In a one-directional pipeline, an error introduced in the requirements stage flows into the plan, from the plan into the tasks, and from the tasks into the code, gathering confidence at every step. Research on multi-agent development pipelines confirms that waterfall-shaped agent workflows allow hallucinations and misreadings from early stages to accumulate downstream, producing software that is internally consistent and externally wrong. The pipeline does not merely fail to catch the error. It launders it.

## 2.3 The misreading of Royce

It is worth being precise about history, because the historical argument is frequently deployed carelessly on both sides of this debate.

Winston Royce's 1970 paper, commonly cited as the origin of waterfall, is doubly misread. Royce presented the now-famous sequential diagram as a description of a process that "is risky and invites failure," and immediately proposed corrections, including iteration between adjacent phases and building a pilot version first. But the correction most relevant here is his prescription of massive documentation as risk control: on the order of fifteen hundred pages of specification for a five million dollar project. That documentation was **evidentiary**. It existed to prove that analysis had occurred before construction, to satisfy contractual and audit requirements. It was written once and read rarely, if ever. Nothing executed against it. Its truthfulness had no operational consequence.

The Agile Manifesto's declaration of "working software over comprehensive documentation" was a rebellion against exactly that artifact class: documents whose production consumed the project's attention and whose staleness cost nothing, because nothing depended on them.

This historical framing matters because it defines what has and has not changed. The manifesto never devalued documentation as such. It devalued documentation with no operational role. As Part II will argue, agentic systems have created, for the first time, a class of documentation with a direct operational role, and this changes the economics of the manifesto's tradeoff without invalidating its logic.

## 2.4 What agile actually was: the backward edge

A common objection, sometimes offered even by agile sympathizers, is that iterative development is “just repeated waterfall”: each sprint contains analysis, design, construction, and test, so the difference is merely one of granularity.

This objection misidentifies waterfall’s defining property. Waterfall is not defined by the sequence of activities. Every act of building anything involves deciding, doing, and checking, in roughly that order. Waterfall is defined by the **freezing of outputs between phases**: the requirements document, once approved, becomes an immutable input to design; the design, once approved, becomes an immutable input to construction. Learning cannot flow backward. Discoveries made during construction have no sanctioned path back into the requirements.

Agile’s essential innovation is the **backward edge**: iteration N’s discoveries renegotiate iteration N+1’s scope and can invalidate parts of the standing plan. The demo exists to generate backward-flowing information. The retrospective exists to apply backward-flowing information to the process itself. A sprint that contains analysis, build, and test but forbids its discoveries from altering the plan is not agile at small scale. It is waterfall at small scale, a failure mode well known enough to have a name (Scrumfall), and it is precisely the failure mode that current SDD tooling reproduces in compressed time: spec to plan to tasks to code, forward only.

The design consequence for aigile is direct: **the backward edges must be first-class citizens of the methodology**. Every protocol defined in Part III (deviation handling, drift management, demo verdicts, retrospectives) is, at bottom, a formalized backward edge: a defined channel through which implementation learning, user feedback, and process learning flow back into the intent layer.

---

# P
cha

Agi
der
spe
con
loc
tea
cod
che
con
and
tha
acc
aut
the
pec
wo
cha
the
con
par
the
pre
bec
one
spe
stru
ele
fra
Par

##

am
me

Cla
cou
“wo
soft
con
doc
bec
tea
acc
kno
con
was
doc
sto
pec
refi
and
soc
kno
alw
for
and
but
fun
sys
trib

Age
stat
bet
ses
con
sta
An
imp
sub
wo
yes
reta
of i
tac
ma
ligh
doc
vial
tea
doe
age
it o
exis
tea

Thi
leg
pro
SDI
solv
hor
mu
“Ag
wit
me
arc
coll

vib
flue
cun
inc
aig
the
arti
def
Sec
dur
tha
con
(pla
tha
cult
sen
eng
me
in h
tea
eph
tha
disp
a d
tha
reg
ratl
mai

It is
stat
exp
bec
dep
con
of a
**tril**
**kn**
**sh**
**tre**
**def**
**for**
**col**
hur
hur
to-a
age
Cor
arc
doc
rec
dec
spe
wer
valu
hur
agil
the
bec
con
was
doc
dec
doe
tole

sha
kno
dep
and
mu
in k
tru
the
Sec
10.

##
Doc
bec
bea

Her
sin
con
eco
cha
des
sta

In t
age
pro
doc
was
**cor**
**ove**
wri
hur
hur
occ
dec
the
sys
wik
mil
wo
stal
imr
doc
dec
uni
the
jud
aga
in i
eco
rati

In a
wo
dur
doc
laye
**age**
**sta**
**wo**
**ses**
**ope**
**inp**
con
arc
con

sto
spe
the
des
the
are
wo
wr
in t
not
con
cau
to c
and
bui
thi

Thi
inc
str
the
the
dis
kee
doc
cur
imr
cos
neg
san
fail
Doc
sto
con
art
bec
**bea**
**inf**

Tw
foll

1. T
ma
dic
ove
refi
“W
sof
con
doc
rem
for
and
doc
nev
cla
**op**
**do**
has
and
“w
sof
the
bec
sof

cor
dep
Bec
laye
bea
be
like
thir
ver
rev
for
(dri
Sec
refa
gar
Sec
kep
its
per
spe
buc
10.

##
eco
ver
wh

##
bot
mo

Wh
gen
nea
ver
bec
con
ind
obs
may
vel
dra
inc
cod
spe
wit
inst
gro
tec
and
rev
The
dist
bet
**ver**
the
wha
inst
said
**val**
the
wha
act
Age
inc
cap

firs
is d
hur
dev
in S

##
fail
eco
mo

A fr
con
cur
is t
hav
lazy
age
Thi
und
and
Rul
is n
cha
fail
rati
res
eco
asy
whe
is fi
rev
exp
rev
skip
Exh
not
eco
Str

Aig
ado
**ec**
**rev**
**dis**
des
Cor

- **Ex**
**acc**
**cri**
mu
spe
pos
exp
tes
rev
int
whe
the
Ver
the
me
hur
the
ma
vali

ans
**not**
Wh
spe
cha
rev
spe
as t
cod
par
eve
rea
arti
wha
**Ato**
**req**
diff
spe
tog
rev
Upo
spe
not
tasl
the
dor
**inc**
**siz**
incr
cor
whe
can
one
one
imp
wit
ses
con
sing
alig
con
usu
eac
hur
spa
con
and
rad
mis
aig
rep
sto
ritu

##
lea
con
buc

In c
the
loo
und
the
hea
by-

imp
Wh
imp
by-
dis
hur
mo
sys
ove
wit
qua
fee
is t
the
fra
dep

Aig
hur
con
as
res
del
ma
rat
ass
me
app
thr
III:
vali
(Se
par
ret
tha
reb
sha
mo
11)
doc
lay
so
nev
hur
can
acc
mo
sys
fra
doe
on
con
will
its
qua
mo
not
the
alr

# Part III: The aigile framework

# 6. The core invariant and the three-layer artifact model

## 6.1 The core invariant: humans own the why

Every configuration of aigile, from pair collaboration to fully agentic delivery, preserves one invariant:

> **Humans always own intent. Automation only ever moves who owns the check.**

Intent enters the system in two forms: **innovation input** (what the product should do and for whom, expressed as feature intent) and **architectural authority** (the technical constitution: constraints, invariants, and foundational decisions). In collaborative setups, humans additionally perform much of the verification. In fully agentic setups, verification is largely delegated. In no setup is intent delegated, because an agent defining what "right" means would be grading its own exam question. This invariant is what makes aigile one framework with a dial rather than two frameworks with a gap; the dial is specified in Section 13.

## 6.2 The three layers

All project artifacts belong to exactly one of three layers, distinguished by their rate of change, their maintenance model, and their authority.

**Layer 1: The durable layer (maintained deliberately).** The constitution (Section 7), architecture decision records, domain invariants, cross-cutting concepts, and collaboration norms. This layer plays the role that culture and senior memory played in human teams. It changes slowly, through a deliberate amendment process. It is read by every agent in every session and by every human onboarding. It is the most expensive layer per line and must therefore be the smallest.

**Layer 2: The ephemeral layer (cheap and disposable).** Feature intents, story specifications, task descriptions, and their acceptance criteria. These artifacts are written to be condensed but fully comprehensible to a human reviewer: small enough to review in one sitting, complete enough that the reviewer can verify the right thing is being built. A story spec should fit on one screen. Once an increment is validated and sealed, its ephemeral artifacts have served their purpose; they are archived, not maintained.

**Layer 3: The derived layer (regenerated, never maintained).** Descriptions of how the system currently works: module overviews, API behavior summaries, data flow explanations. In aigile these are not hand-maintained documents. They are generated on demand by an agent reading the code and tests, timestamped, and treated as disposable. A document that is regenerated from the source of truth every time it is read cannot drift. Roughly half of what traditional projects maintained as documentation belongs in this layer, and moving it there is one of the largest single reductions of drift surface available.

> **Figure 1 (schematic).** Layer 1 (constitution, decision records, invariants) constrains Layer 2 (feature intents, story specs, tasks), which drives the code and executable tests. The derived Layer 3 (module overviews, behavior summaries) is regenerated from code on demand. Backward edges: implementation learnings flow from code back into Layer 2; promotions and amendments flow from Layer 2 into Layer 1.

*Figure 1: The three-layer artifact model. Solid arrows are the forward flow; dashed arrows are the backward edges that make the system agile rather than waterfall.*

### 6.3 Layered truth

The layers also settle, in advance, the question of authority when representations disagree:

- For **what the system does**, running code and its executable tests are the truth. Tests are the executable portion of the specification; when prose and tests disagree, the prose is corrected. This is "working software over documentation" applied with precision.
- For **what the system should do and why**, the durable layer is the truth, because intent cannot be encoded in tests. When code violates the constitution, the code is the defect.
- When these two truths contradict each other, the contradiction is not a documentation problem. It is a discovered defect in one of them, and it is information: a free bug report. Section 10 defines the protocol.

## 7. The constitution as versioned law

The constitution is the anchor artifact of the durable layer: the project's architectural principles, binding constraints, technology decisions, quality bars, and collaboration rules. Because every agent reads it in every session, an error in it propagates instantly and everywhere. It therefore requires a governance model stronger than ordinary documents and more responsive than waterfall baselines: **law-like process, Git-native mechanics**.

- **Amendments are pull requests** against the constitution, carrying a mandatory rationale in decision-record form: what changes, why, which alternatives were rejected, what is invalidated.
- **Approval follows the authority model** (Section 13): a competent human holds final authority in collaborative setups; a designated lead-architect role, which may be an agent in fully agentic setups, holds it there, with humans retaining the appointment of that role.
- **Versions are semantic.** Patch for clarification, minor for a new constraint, major for anything that invalidates existing specifications or code.
- **Every increment records the constitution version it was built under.** This one line of metadata purchases three capabilities: auditability ("this module predates the event-sourcing amendment"), mechanical impact analysis after a major amendment (list everything built under older versions and decide migration versus grandfathering), and honest rollback (reverting an amendment does not undo code built under it, but the version trail identifies exactly which increments carry the reverted assumption).
- **The constitution is refactored, not only extended.** Section 11 defines the gardening rule that prevents it from growing monotonically into an unreadable statute book.

The constitution is deliberately hard to change and deliberately possible to change. It reacts to allowed inputs through a defined channel, exactly as a healthy architecture review process does in a strong human organization. What it never does is silently drift or silently freeze.

# 8. The working loop: intent, slice, build, validate

The day-to-day shape of aigile is a short loop, and everything else in this paper exists to keep this loop honest.

1. **Intent.** A human expresses feature intent. Agents may assist in drafting, structuring, and challenging it, but the human owns it.
2. **Slice.** The feature is decomposed into stories and tasks (as most SDD frameworks already do), each sized by the increment rule: one human sitting to review, one agent session to implement. The first slice of any feature is a walking skeleton (Section 12).
3. **Specify thinly.** The slice receives a condensed specification and executable acceptance criteria. Specification detail is budgeted (Section 10.1): describe what must be true, not what happens to be true.
4. **Build.** An agent (or human-agent pair) implements the slice. Deviations from spec are handled by the protocol in Section 9, never silently.
5. **Verify.** Acceptance tests run mechanically. The atomic PR (code-diff, test-diff, spec-diff) is reviewed as one unit.
6. **Validate.** At defined points (Section 12), humans put brains, eyes, and hands on the running increment.
7. **Learn.** The retrospective (Section 11) routes what was learned into the correct layer, and the loop repeats.

Progress is deliberately kept small at every step. Small pull requests keep human review feasible; small slices keep agent context windows coherent; small demos keep validation cheap. Smallness is not a stylistic preference in aigile. It is the load-bearing property that keeps every feedback edge short.

# 9. The deviation protocol

Agents and humans mid-implementation regularly discover that the specification is wrong, incomplete, or self-contradictory. This is not a failure of the process. It is the process working: implementation is where specification error becomes visible. What matters is the protocol, because the two naive answers both destroy the framework. Always stopping recreates waterfall's change-control paralysis. Always guessing recreates vibe coding.

## 9.1 Observable classification

The deviation is classified by an **observable** boundary, not by judgment, and specifically not by the self-assessment of the party that wants to deviate (an agent is a poor judge of whether its own shortcut has behavioral consequences, and letting it judge builds a conflict of interest into the protocol):

> A deviation is **behavioral** if any acceptance test, public interface, or constitution clause must change to accommodate it. If all tests still pass and no contract moves, it is an **implementation detail**.

## 9.2 Tiered response

- **Implementation details** proceed without stopping, but are logged and batch-reviewed at the retrospective. The log is not bureaucracy; it is the raw material for detecting spec sections that chronically over-specify (Section 11).

- **Behavioral deviations pull the andon cord.** Work on the story stops. A consortium convenes: in human-centric setups, the system recommends options and a competent human decides; in fully agentic setups, a lead-architecture agent may hold this authority within its constitutional mandate, with humans retaining the mandate itself. The consortium operates under a **latency budget** (for example: a decision within the hour, with a defined default if no objection is raised). Without the latency budget, the consortium degrades into a change control board, and the framework quietly reinvents the governance sludge agile escaped.
- **Constitutional conflicts** (the deviation reveals that the constitution itself is wrong or self-contradictory) escalate to the amendment process of Section 7. They never get patched locally.

Every accepted behavioral deviation produces its spec-diff in the same pull request as the code that embodies it. Divergence is resolved at the moment it is born, which is when it is cheapest.

# 10. Drift management

Even with the deviation protocol, representations of a system diverge over time. Hotfixes enter sideways, assumptions decay, the world changes. Aigile's position is that **drift is thermodynamics, not sin**. Methodologies that treat drift as shameful get hidden drift. The goal is not zero drift but a short **drift half-life**: minimal time between divergence appearing and being detected, classified, and resolved. Drift is managed exactly like defects, through a four-stage pipeline.

## 10.1 Prevent

- **Atomic PRs** (Section 5.2) close the main channel through which drift is born.
- **Executable acceptance criteria** shrink the drift-capable surface: a test cannot drift silently, because continuous integration screams.
- **The spec detail budget.** Every line of prose specification is a standing drift liability. Specify at the level of abstraction that changes slowest: what must be true, never what happens to be true. "Orders must survive process restart and be queryable by customer within 500 ms" may hold for years; "orders are stored in PostgreSQL table `orders` with columns x, y, z" will drift within weeks. Over-specification is not diligence. It is the manufacture of future drift.
- **The derived layer** (Section 6.2) removes roughly half the traditional documentation corpus from the drift surface entirely, because regenerated documents cannot drift.

## 10.2 Detect

- **The conformance function: a linter for the map.** Something must periodically read the specification layer against code and tests and flag contradictions ("spec section 4.2 claims idempotent retries; the implementation retries without deduplication"). The conformance function only ever produces findings, never autonomous fixes.
- **Freshness metadata.** Each durable and ephemeral spec section carries the commit hash at which it was last verified, making staleness measurable: a spec-freshness metric analogous to test coverage. Drift stops being ambient anxiety and becomes a number.
- **Scaling the function.** In small teams the conformance function is a human-led **ritual**, not a role: the retrospective doubles as a

fifteen-minute conformance sweep, agent-assisted on demand (“read spec section X against the current module and list contradictions”). This deliberately avoids assigning humans a standing vigilance task, a class of work humans measurably perform badly. The **graduation trigger** is defined by evidence, not team size: when the ritual sweep regularly surfaces more than trivial descriptive drift, the spec surface has outgrown the ritual, and a standing conformance agent pays for itself. The methodology names the trigger; teams choose the tooling.

### 10.3 Triage

Classification reuses the observable boundary of Section 9, because drift is deviation discovered late. The question is mechanical: what must change to resolve it?

| Drift class | Definition | Resolution path | Ceremony |
|---|---|---|---|
| **Descriptive** | Code is right; spec describes it wrongly. Only prose must move. | Conformance function proposes a spec-diff; human batch-approves at the retro. | Low |
| **Normative** | Code violates documented intent. Tests or interfaces must move. | A defect story is created; constitution violations pull the andon cord. | Medium |
| **Constitutional** | Truth layers contradict each other, or constitution clauses conflict. | Escalates to the amendment process (Section 7). | High |

### 10.4 Learn

Drift patterns are design feedback about the specification layer itself. A section that drifts repeatedly is saying one of two things: it is over-specified and should be demoted to the derived layer or deleted, or it expresses something genuinely fundamental that keeps being violated and should be **promoted** to the constitution, where it gains enforcement teeth. Per-section drift frequency is the signal that drives the gardening rule of Section 11.

## 11. The retrospective and the gardening rule

After each successfully validated feature, humans and agents hold a retrospective. It has three distinct outputs, and routing them correctly is the entire trick.

1. **System learnings** (“this module actually behaves like X,” “this integration has a hidden constraint”) flow into specs, decision records, and, when fundamental, constitutional amendment proposals.
2. **Collaboration learnings** (“this spec wording confused the

agent," "this task slice exceeded one session," "this prompt pattern worked") flow into the durable process layer. This category is genuinely novel. Human teams never wrote down how to phrase things so a colleague understands, because colleagues learned. Agents do not learn between sessions, so collaboration knowledge must compound in artifacts or it evaporates every day. The retrospective is also the primary mechanism by which participating humans keep their mental model of the system current: the comprehension budget of Section 5.3, spent deliberately.
3. **Drift findings** from the conformance sweep (Section 10.2) are batch-triaged.

**The gardening rule.** Retro outputs accumulate monotonically by default. Twenty features in, an ungoverned constitution is a bloated statute book of amendments, half obsolete, consuming agent context and human patience. Therefore, on a defined cadence (for example every Nth retrospective), the retrospective is a **gardening session**: lessons are consolidated into principles, superseded clauses are deleted, chronically drifting spec sections are demoted or promoted per Section 10.4. Lessons are composted into principles, not stockpiled as incidents. The durable layer is refactored with exactly the seriousness that code is, because it is load-bearing (Section 4).

# 12. The validation architecture

Verification asks whether the thing was built right. Validation asks whether the right thing was built. Aigile's position is that validation is structurally human at every point on the automation dial, for a definitional reason: "right" is constituted by human and user intent, and a system validating its own rightness is grading its own exam question.

## 12.1 Three demo triggers

Validation happens at demos of running software. A single demo gate at feature completion would quietly recreate mini-waterfall at feature scale: all increments built, intent checked at the end, misdirection discovered when it is most expensive. Aigile therefore defines three triggers.

- **The skeleton demo.** The first increment of any feature must be a walking skeleton: a thin vertical slice, end to end, ugly and minimal, demonstrated to a human before the remaining increments proceed. This is where misunderstood intent is cheapest to catch, and it costs minutes.
- **The feature demo.** The guaranteed gate at feature completion: a full acceptance walkthrough with hands on the running software. This is the formal validation moment and the smallest unit of demo cadence.
- **Risk-pulled demos.** Any increment that touched a constitution clause, took the andon path, or landed in an area with poor drift history receives a human look on demand. Everything else flows through on mechanical verification alone.

The result is a human attention budget that scales with risk and novelty rather than with volume, which is the only way the fully agentic end of the dial remains honest without drowning the humans it kept.

## 12.2 The verdict as artifact

A demo verdict is an artifact with a defined destination, never a feeling:

- **Validated.** Acceptance criteria are marked human-confirmed; the increment is sealed.
- **Misaligned.** A scribe agent drafts spec-diffs directly from the demo feedback; the human approves them; the delta becomes stories. This is the framework's formal channel for re-feeding user and stakeholder feedback into the specification set, so that documentation stays current for every future agent session and every future human onboarding.
- **Intent evolved.** The demo revealed that the humans want something different from what they asked for. This is not a defect. It is a discovery, and it routes into the feature-intent backlog, possibly with a constitutional note.

### 12.3 The brains-eyes-hands rule

A validation demo is invalid if the human only watched a recording or read a summary. Agents may prepare the environment, seed the data, and script the walkthrough, but during validation the hands on the keyboard are human. The entire epistemic value of the gate is an unmediated human encountering the actual artifact. The moment an agent summarizes the demo for the human, the layer of mediation the gate exists to remove has been reinserted, and demo theater follows. Owning intent carries this responsibility with it; validation by one's own brain, eyes, and hands is not overhead on ownership, it is what ownership physically consists of.

## 13. The authority matrix and graduated autonomy

### 13.1 One dial, not two modes

Aigile is often easiest to explain as two setups: a **human-AI collaboration mode**, in which both parties actively work on the system and a competent human holds defining authority supported by AI input, and a **fully agentic mode**, in which agents hold the operational roles and humans contribute innovation input and lead architectural expertise. Both descriptions are accurate, but they are not two methodologies. Under the core invariant (humans own the why in both), the difference reduces to **who owns each check**, and that is not one switch but a matrix: an authority setting per verification layer.

| Verification layer | Collaborative end | Fully agentic end | Can it move right? |
|---|---|---|---|
| Constitution amendments | Human decides, AI advises | Lead-architect agent within a human-granted mandate | Partially; mandate stays human |
| Feature intent and validation | Human | Human | **Never** (definitional) |
| Spec conformance (drift) | Human ritual, agent-assisted | Standing conformance agent | Yes, on the graduation trigger |
| Behavioral verification (review) | Human reviews every atomic PR | Agent review; human sampling | Yes, with evidence |
| Mechanical | | | |

| | | | |
|---|---|---|---|
| verification (tests, CI) | Automated | Automated | Already right |

A project does not choose Mode A or Mode B. It has an authority matrix, and “fully agentic” simply means most rows have moved right. This framing yields a migration story a binary cannot: teams start collaborative and **earn** autonomy, layer by layer.

### 13.2 Graduated autonomy

Autonomy is granted on evidence and revoked on incident, in the manner of graduated licensing. A verification responsibility moves from human to agent only after a defined track record, for example N consecutive increments in which human review found nothing the agent’s checks had missed. An incident moves the row back left until the record is rebuilt. This gives organizations an answer to “how do we trust this” that is better than vibes: a paper trail of earned delegation.

### 13.3 Correlated verification failure

Human peer review works partly because two humans have different brains; their errors are weakly correlated. A builder agent and a reviewer agent sharing the same base model share blind spots, so agent-on-agent review can be confidently and **consistently** wrong. Fully agentic configurations must therefore make verification architecturally independent from generation:

- Use a different model family for review and conformance roles than for builder roles.
- Frame reviewers adversarially (“find the ways this violates the specification”), not confirmatively (“check this”).
- Prefer property-based and executable checks, which do not depend on model judgment at all, wherever they can carry the load.

This requirement belongs in the constitution of any agentic setup, not in a best-practices appendix.

## 14. The self-improvement circle and its external anchor

At the fully agentic end of the dial, the retrospective becomes a recursive self-improvement circle: agents measure the process, propose improvements to specs, prompts, task sizing, and tooling, and apply them. This is aigile’s most powerful configuration and its most dangerous one, and the framework must say both things in the same breath.

A loop in which agents improve the process against measurements agents themselves define is a Goodhart machine. It will optimize its proxies (drift counts down, cycle time down, test suites green), and the proxies will smoothly decouple from what the humans actually care about. Every metric-driven system does this. An autonomous one does it faster and without embarrassment.

The circle is safe only if its ground truth is injected from outside the loop, and the framework already contains the injection point: **the human demo verdict of Section 12 is the external grounding signal**. The circle may self-optimize on efficiency without limit, but its fitness function, “was this actually right,” enters exclusively through human brains, eyes, and hands at the validation gates. The demo is therefore not a legacy ritual that survives into the agentic mode out of

nostalgia. It is the anchor that makes the self-improvement circle safe to run at all. The two mechanisms are one design, and they must be adopted together or not at all.

This closes the argument of the paper in a satisfying way: at maximum automation, the one loop that structurally cannot close without a human in it is precisely the iteration-demo-feedback loop, which is to say, the agile core. Aigile's claim that agile survives the agentic era is not sentiment. It falls out of the architecture.

---

# Part IV: Minimum viable aigile

A methodology that cannot state its own minimum form gets adopted at maximum weight and blamed for the result. The following five rules constitute minimum viable aigile: adoptable by a two-person team with one agent, using nothing beyond a Git repository and a folder of markdown. Everything else in Part III is machinery added when evidence (not fashion) demands it.

1. **Write a one-page constitution and treat it as law.** Architectural constraints, non-negotiable invariants, quality bar. Change it only by pull request with a written rationale. Stamp increments with its version.
2. **Slice small: one sitting, one session.** Every unit of work must be reviewable by one human in one sitting and implementable by one agent in one context window. The first slice of any feature is a walking skeleton, demoed before the rest proceeds.
3. **Ship the spec-diff in the PR.** Code, tests, and specification changes travel as one atomic unit. Acceptance criteria are executable wherever possible. The spec describes what must be true, not what happens to be true.
4. **Stop on behavioral deviation.** If an acceptance test, public interface, or constitution clause must change, work stops and a human decides, fast. Everything else proceeds and is logged.
5. **Validate with brains, eyes, and hands, then garden.** A human operates every feature before it is sealed; the verdict produces spec-diffs or new intent, never just a feeling. At a regular cadence, the retrospective prunes and promotes the document layer instead of only growing it.

Each rule is the seed of its full-framework counterpart (Sections 7, 8, 5.2, 9, and 11 through 12 respectively), which is the point: scaling up in aigile means elaborating rules the team already lives by, never replacing them.

---

# Part V: Relationship to current SDD practice

Aigile is not a rejection of spec-driven development tooling. Tools such as Spec Kit and Kiro supply useful mechanics: constitution and steering files, structured decomposition, task templates. Aigile can be understood as a corrective theory of use for that tooling:

| Dimension | Prevailing SDD practice | Aigile |
|---|---|---|
| Process shape | Forward pipeline: spec, plan, tasks, code | Loop with first-class backward edges |

| | | |
|---|---|---|
| Spec size | Comprehensive up front | Budgeted; thin ephemeral layer, minimal durable layer |
| Source of truth | The specification | Layered: code and tests for behavior, constitution for intent |
| Documentation model | Maintained corpus | Durable (maintained), ephemeral (disposable), derived (regenerated) |
| Drift | Treated as discipline failure | Managed like defects; measured half-life |
| Review | Human reads generated documents | Review economics: diffs, executable criteria, sized increments |
| Validation | Implicit in spec conformance | Explicit human gates; brains-eyes-hands rule |
| Autonomy | Binary (assistant or autopilot) | Authority matrix with graduated, evidence-based delegation |
| Self-improvement | Not addressed | Retro circle, externally anchored by demo verdicts |

The one-sentence summary of the difference: **prevailing SDD tries to make the specification good enough that feedback becomes unnecessary; aigile makes feedback cheap enough that the specification can stay small.**

---

# Part VI: Open questions and research agenda

This paper is foundational and therefore incomplete by design. The following questions are open and are the intended subjects of subsequent papers.

1. **Empirical validation.** No rigorous comparative data yet exists for SDD variants, iterative prompting, or aigile. Controlled studies should compare defect rates, rework rates, drift half-life, and human hours per validated feature across process shapes, on both greenfield and brownfield systems.
2. **Metrics.** Candidate aigile-native metrics require definition and calibration: drift half-life, spec freshness, human validation hours per feature, autonomy track records per authority row, and comprehension measures for the human participants.
3. **The comprehension budget.** How much hands-on contact does a human need to remain a competent validator of a system they no longer implement? This is an empirical question with major staffing implications.
4. **Multi-team and multi-agent scaling.** How do constitutions federate across teams? What is the equivalent of an interface contract between two aigile loops?
5. **Regulated environments.** The constitution-version stamping and

verdict artifacts appear naturally auditable; mapping them onto specific regulatory regimes (medical, automotive, financial) is future work.

6. **Tooling.** The conformance agent, freshness metadata, spec-diff review surfaces, and verdict capture all want first-class tool support. This paper deliberately specifies triggers and protocols rather than tools.

---

# Glossary

- **Aigile.** The methodology described in this paper: agile principles re-derived for human-AI collaborative and fully agentic software engineering.
- **Andon.** The stop-the-line mechanism triggered by behavioral deviations, named for the Toyota Production System cord.
- **Atomic PR.** A pull request containing code-diff, test-diff, and spec-diff as one reviewable unit.
- **Authority matrix.** The per-layer assignment of verification ownership between humans and agents.
- **Backward edge.** Any formal channel through which learning flows from implementation, users, or process back into the intent layer.
- **Brains-eyes-hands rule.** Validation demos require unmediated human operation of the running software.
- **Comprehension budget.** Deliberate expenditure of human time to keep human mental models of the system accurate.
- **Conformance function.** The ritual or agent that reads specifications against code and tests and reports contradictions.
- **Constitution.** The durable, versioned statement of architectural principles, constraints, and collaboration law.
- **Derived documentation.** System descriptions regenerated on demand from code and tests, never maintained by hand.
- **Drift half-life.** The characteristic time between a divergence appearing and being detected, classified, and resolved.
- **Gardening rule.** The scheduled consolidation, promotion, demotion, and deletion of durable-layer content.
- **Graduated autonomy.** Evidence-based delegation of verification responsibilities from humans to agents, revocable on incident.
- **Increment sizing rule.** One human sitting to review; one agent session to implement.
- **Spec detail budget.** The discipline of specifying what must be true rather than what happens to be true, because every line of prose is a drift liability.
- **Velocity paradox.** Rising code production speed coexisting with rising instability and debt when verification and validation do not keep pace.
- **Walking skeleton.** The mandatory first slice of a feature: a thin, end-to-end, demonstrable vertical cut.

---

# Appendix A: Historical note on sources of the argument

The reading of Royce (1970) in Section 2.3 rests on the paper's own text: the sequential diagram is introduced as risky and failure-prone, with iteration, piloting, and heavy evidentiary documentation offered as mitigations. The Agile Manifesto (2001) and its principles supply the value hierarchy that Section 4 refines rather than overturns. The critique of heavy SDD draws on publicly reported practitioner

evaluations of Spec Kit and comparable tools during 2025 and 2026, including order-of-magnitude slowdowns relative to iterative prompting, and on research findings that waterfall-shaped multi-agent pipelines accumulate early-stage errors downstream. The andon and gardening concepts adapt, respectively, the Toyota Production System's stop-the-line practice and the long-standing software practice of refactoring, applied here to the document layer. Graduated autonomy adapts graduated licensing. None of these borrowings is decorative; each imports a tested mechanism into a place where the agentic setting recreates the original problem.

---

*End of working paper v0.1. This document is intended as the stable foundation for: (1) in-depth methodology papers per Part III section, (2) a concrete playbook with worked examples, (3) presentation material, and (4) website content. Terminology defined in the glossary should be treated as canonical for derivative documents.*