Integral: From Theory to Node to Network

A Development Guide for Contributors of All Kinds

Version 0.1

Preface

What This Document Is

This guide exists at the intersection of two realities: a detailed theoretical architecture and the practical work of building it.

The *Integral Technical White Paper (v0.1)* describes a federated, post-monetary cooperative economic system — five interdependent subsystems, dozens of modules, formal specifications, mathematical sketches, and pseudocode. It is a serious and rigorous attempt to specify what a non-market economic coordination system could look like at scale. If you have not read it, it is available in the project repository and serves as the primary reference for everything described here.

This document is not a replacement for the white paper. It is a proposed bridge between that vision and the first real lines of code, the first working node, and the first community of contributors building toward it. Where the white paper asks *what should Integral be*, this guide asks *how do we begin building it* — and more importantly, *how do we begin building it honestly*.

That word — honestly — carries specific weight here. It means acknowledging from the outset that the white paper is not a finished blueprint. It is a first-pass specification of extraordinary ambition, written at a level of abstraction that necessarily leaves many critical questions open. Some of those questions are technical. Many are philosophical. Some concern whether the system's proposed mechanisms will actually produce the outcomes they are designed for when real people use them under real conditions.

None of those questions are fully answered yet.

This project is as much a problem-solving endeavor as it is a coding project. The contributors who join it are not implementers executing a settled design. They are participants in an ongoing, circular process of critical evaluation, principled experimentation, and honest revision. The white paper is the starting point for that process, not the conclusion of it.

The Central Challenge

Integral proposes to do something that has never been done at scale: replace market coordination — with all its failures and structural injustices — with a democratic, ecologically grounded, cybernetically coherent alternative. The ambition is not in question. The difficulty is.

Markets, for all their destructiveness, solve certain coordination problems efficiently. They aggregate dispersed information through prices. They create incentives for production without central direction. They handle scarcity signals automatically. Integral proposes to replace these functions with deliberate democratic governance, open design commons, contribution accounting, and feedback-driven monitoring. Whether those replacements work as well in practice as they do in theory — and under what conditions, at what scale, with what kinds of people — is unknown.

But there is a prior question that reframes everything else.

Market economics is not a stable system with correctable flaws. It is a self-reinforcing loop: capital accumulates, ecological resources are consumed to generate more capital, inequality deepens, the political capacity to regulate the system is steadily eroded by the power of those who benefit from it, and the loop continues. The trajectory is not toward equilibrium. It is toward ecological collapse and the kind of concentrated, entrenched inequality that forecloses the possibility of democratic correction. These are not predictions about a distant future. They are descriptions of processes already well underway.

If that diagnosis is correct — and the empirical case for it grows stronger every decade — then the question is no longer *does Integral work well enough to be worth trying*. The question becomes: *we need a new system, and we need it to be taking root now, because the existing one is destroying the conditions that make any new, better system possible*. The deficiencies in any given proposal are real and must be taken seriously. But they must be taken seriously in the context of what the alternative is. Waiting for a perfect system is not a neutral position. It is a choice to remain inside a system that is actively foreclosing the future.

This reframes how to hold the problems described in this document. They are not reasons for doubt about whether to proceed. They are the specific work that needs to be done. Every unresolved question about ITC weighting, every open problem in federated governance, every tension between democratic legitimacy and response speed — these are problems to be solved *urgently and in practice*, by real communities building real nodes, not problems to be resolved in theory before anything begins. The urgency is not rhetorical. It is structural. The window in which a transition of this kind remains possible is not indefinitely open.

This is not a reason to abandon the project. It is a reason to approach it with rigorous honesty about what has been figured out and what has not — and with the seriousness that the stakes demand. The following section names some of the hardest open questions directly. It is not exhaustive. It is an invitation to think carefully before building, and to keep thinking carefully throughout.

White Paper: Known Hard Problems

The challenges described here are not afterthoughts. They are structural tensions embedded in the architecture itself — places where the white paper's proposed mechanisms may not fully resolve the problems they are designed to address, or where the right solution remains genuinely contested. Every contributor should be familiar with them. Several of them may require revisions to the white paper before the relevant systems are built.

Democratic deliberation and cognitive overload. The CDS is asked to govern everything: production mandates, ecological constraints, contribution weighting policy, certification standards, federation agreements. At small scale, within a tight community, this may be manageable. As nodes grow and federate, the volume of decisions requiring genuine deliberation could overwhelm participants' capacity and time. One proposed approach — borrowed from industrial control theory — is to separate *boundary-*

setting (which requires deliberation) from *within-boundary optimization* (which can be automated). CDS would set the envelope; algorithms would operate within it; CDS would only re-engage when the system hits a hard boundary or fails to stabilize. Whether this adequately preserves democratic legitimacy, or quietly shifts real power to whoever designs the algorithms, is an open question.

The value of labor and the one-to-one question. The white paper proposes weighting labor contributions by skill, scarcity, and ecological sensitivity, with time decay applied to accumulated credits. Both mechanisms have a coherent rationale. Both also introduce significant complexity, governance surface area, and philosophical tension. Traditional time banks operate on a simple one-to-one principle — one hour of any person's time equals one credit — as a deliberate moral commitment to the equal worth of human effort. The argument that differential weighting reproduces market wage logic inside a system designed to transcend it is serious and not fully resolved. Whether weighting and decay are necessary sophistications or unnecessary complications that import the assumptions they were meant to replace is one of the most important questions this community needs to work through before building the ITC system.

It is also worth clarifying a commonly misread aspect of the ITC scope boundary. The white paper's exclusion of democratic participation and creative ideation from ITC generation is not a statement that intellectual work is less valuable than physical labor. The operative criterion is necessity and node mandate: work that the node has identified as operationally necessary through its CDS process — whether physical or intellectual — generates ITCs. A contributor doing mandated research, solving a critical design problem, or performing essential analytical work earns credits in exactly the same way a contributor doing physical fabrication does. Routine governance participation and unsanctioned speculative exploration do not. This distinction is deliberate and coherent, not an oversight.

Verification and the oracle problem. The system's integrity depends on accurate data entry across all five systems. Labor events must be honestly logged. Ecological assessments must reflect reality. Production outputs must be accurately reported. The white paper relies heavily on peer verification and community trust. But strategic misrepresentation — not outright fraud, but selective emphasis, optimistic self-assessment, or gaming verification thresholds — is a persistent risk in any self-reported system. One promising approach is physical triangulation: if the transport cooperative logged fifty tons of steel arriving at your node, your claim to have used ten tons is immediately suspect without requiring any accusation. Outcome-based verification — authenticating labor by confirming that the outputs exist and passed quality assessment, rather than by counting hours — is similarly robust. These approaches need to be designed into the system's core architecture, not treated as edge-case auditing tools.

Governance velocity vs. crisis response. Democratic deliberation takes time. Physical crises — a supply chain collapse, an ecological threshold crossed suddenly, infrastructure failure — do not wait for the CDS pipeline to complete. Stafford Beer's Viable System Model provides a relevant principle: operational units should have autonomy to respond to local shocks immediately, with higher-order deliberation reviewing and adjusting the response after the immediate crisis is contained. Integral needs pre-approved emergency response protocols that activate automatically under defined conditions, with clear sunset mechanisms so that emergency powers don't persist beyond their intended scope. The design of those protocols — what triggers them, who controls them, and how they dissolve — is unresolved.

The transition generation. The people who will build and inhabit the first Integral nodes have been formed entirely within market institutions. They carry market-shaped intuitions about fairness, reward, status, and value — intuitions so deeply embedded that they often feel like descriptions of reality rather than artifacts of a particular economic system. Integral requires not just different software but a different way of relating to work, contribution, and access. The system's interfaces and governance processes need to make their own logic visible and legible enough that people can actually inhabit them — not just comply with them. A CDS that feels like a black box producing outputs, an ITC balance that feels arbitrary, a COS interface that feels like enterprise project management imposed from above: these are not just UX problems. They are adoption failures. The first implementations need to convince people that this works and that it is theirs, not just demonstrate that it runs.

Enforcement without coercion. Integral has no police force, no legal enforcement mechanism internal to the system, and no way to compel compliance with its governance decisions. The proposed enforcement mechanism is network interdependence: a node that violates shared ecological constraints or federation agreements faces the cessation of electricity transfers, material provisioning, tool access, and software updates from the rest of the network. This is coherent at mature network scale, when nodes are genuinely interdependent. In the early network, when nodes may still have substantial market-economy fallbacks through their Interface Cooperatives, the mechanism may lack teeth. The line between a rogue node and a node with a principled disagreement

also needs to be carefully defined — this is a first-order governance problem, not a technical one.

The Interface Cooperative as primary structure, not edge case. The white paper describes Interface Cooperatives as "the primary metabolic bridge" between the existing market economy and Integral during transition. For the Transition Generation — the first real-world nodes — the IC is not optional scaffolding or a risk to be managed. It is the primary legal and operational structure through which a node acquires tools, pays taxes, complies with labor law, and interacts with external supply chains. Land, machinery, utilities, taxes, raw materials, and legal compliance all remain priced and enforced externally. Without a functioning IC, early nodes face a choice between isolation from essential resources or premature legal confrontation with entrenched institutions. Neither is viable. The drift risk is real — the history of cooperatives that began with strong principles and gradually absorbed market logic is extensive — and structural safeguards are necessary: irrevocable asset trusts, sortition-based selection, mandatory rotation, and hard-coded dissolution triggers. But these safeguards must be specified within a fully articulated Minimum Viable IC design, which is a critical Phase 3 or Phase 4 deliverable. This project must produce a practical IC specification before real-world node deployment is attempted. Treating the IC as peripheral understates what early nodes will actually depend on.

Legal and jurisdictional reality. Integral nodes will exist within states that have property law, labor law, tax law, and regulatory frameworks designed around market institutions. A CDS decision that violates local labor law is unenforceable and exposes participants to legal liability. The white paper is explicit that taxation and regulatory compliance remain primary constraints throughout the transition — this is not a concession of principle but a strategic necessity. The relationship between Integral's internal governance and external legal systems requires careful navigation from the beginning of node development, not resolution in a later phase.

The expertise concentration problem. OAD certification of complex goods — medical devices, structural components, electrical systems — requires genuine technical expertise. Expertise concentrates. The people capable of certifying a load-bearing structural design are few, and their judgment cannot be fully democratized without being diluted into meaninglessness. This creates a de facto authority structure inside OAD that the governance architecture doesn't formally account for. How technical authority is legitimized, constrained, and held accountable within a democratic framework is an open design problem.

None of these problems are fatal to the project. Several have promising partial solutions, described in the relevant sections of this guide. All of them require more thinking, more debate, and in several cases, real-world experimentation before they can be considered resolved. The community that builds Integral will not just be writing code — it will be working through these questions in practice, and feeding what it learns back into the architecture. That process is the project, as much as any deliverable.

The Approach: Starting Simple, Growing Honestly

Given the scale of what is unresolved, the question of where to begin is not obvious. The approach this guide proposes is not to solve everything before building anything, nor to build quickly and treat the hard questions as someone else's problem. It is to start with the simplest possible version of the system that is still architecturally honest — that connects all five systems in a real feedback loop, that enforces the foundational principles in software rather than just in policy, and that generates real operational experience that can inform the harder questions.

Complexity is introduced only when it is earned: when the simpler version has demonstrated its limits, when the data exists to make the more sophisticated mechanism meaningful, and when the community has the shared experience to evaluate it honestly. This discipline — building minimally but faithfully, deferring sophistication without deferring principle — is what this guide means by the phrase *minimalistic but faithful*. It is not a compromise. It is an epistemically honest response to genuine uncertainty.

The most important corollary: **no part of the architecture should be treated as settled until it has been tested**. The white paper's specifications are hypotheses, not axioms. When early implementation reveals that a mechanism doesn't work as described — that a governance process produces gridlock, that a data structure can't represent something it needs to represent, that a principle produces perverse outcomes under real conditions — the response is revision, not rationalization. The document you are reading will itself be revised.

How This Document Is Organized

This guide is written for contributors of all kinds. You do not need to be a developer to find your place here, and you do not need to be a systems theorist to contribute meaningfully to the build.

The document moves from philosophy to architecture to practice. It opens with the development philosophy that governs decisions about what to build first and why. It then describes the five core systems in accessible terms, maps how they depend on one another, and specifies the minimum viable version of each that the early development stage will produce. From there it addresses the data structures and inter-system interfaces that must be established early, and the staged path by which the system grows in complexity over time — from foundational development, through simulation and testing, to real-world deployment and network expansion. It closes with a practical guide to contributing: how the GitHub organization is structured, how decisions are made, how different kinds of contributors engage, and how Discord and GitHub function as complementary coordination tools. The mechanics of that contribution process are described in full in Section 6.

Two things are worth keeping in mind as you read. First, technical sections are written to remain understandable to non-technical readers — if a passage feels opaque, that is a flaw in the writing worth flagging, not a signal that the content is not for you. Second, this document is itself a living artifact. It will be revised as the project develops (see repository), as early assumptions are tested against reality, and as the contributor community grows and finds its own rhythm.

> **New here and not sure where to start?** Go directly to **Section 6 — Contributing to Integral**. It explains who contributes, how to apply, where work happens, and how the project governs itself — in plain language, without requiring you to read the full architecture first. Come back to the earlier sections when you're ready to go deeper.

How to Read This Document Depending on Your Background

Five contributor types are used throughout this guide to help organize entry points into the project. These are not formal roles or classifications — they simply help readers focus on the sections most relevant to their interests and experience.

If you are a Builder — a developer, systems architect, or engineer — the most immediately actionable material is in Sections 3, 4, and 5: the minimum viable module set, the critical data structures, and the staged expansion path. These sections define what must be built first and how the systems connect. Appendix C may also be useful as a conceptual map of software architecture concepts for grounding the technical decisions in this guide. The white paper's pseudocode and formal specifications serve as the primary technical reference alongside this guide.

If you are a Thinker — a systems theorist, economist, governance scholar, ecologist, or domain expert engaging with the architecture at the level of ideas — Sections 1 and 2 are your primary entry point, followed by the white paper itself. The GitHub repositories include issue templates specifically intended for critique, challenge, and theoretical revision. That process is considered as important to the project's integrity as the code itself. Your role is not to validate the architecture but to stress-test it.

If you are a Practitioner — a cooperative organizer, mutual aid worker, or commons practitioner with real experience running non-market coordination — Sections 1, 2, and 6 will likely be most relevant. The architecture described in the white paper draws heavily on decades of practice in mutual aid, cooperative economics, and management cybernetics. What works and what fails in real cooperative environments is knowledge the project cannot derive from theory alone. Your experience is not merely context for the build — it is essential input to it.

If you are a Maker — a writer, designer, communicator, or documentarian — Section 6 describes how your work fits into the contribution structure and where it is most needed. The project requires people who can translate between the technical architecture and the communities who will eventually use it. That translation work is not peripheral. A node that fails because participants cannot understand what the system is doing is as much a documentation failure as a technical one.

If you are an Explorer — drawn to the project but still discovering where you fit — begin with this preface and Section 1, then read the white paper abstract and introduction before returning to Section 2. The `#general` channel on Discord is a good starting point for engaging with the community. There is no expectation that you arrive knowing your role; finding it is part of the process, and the project is designed to support that.

A Note on Language

This project sits at the intersection of technical systems design, cooperative economics, cybernetics, and political theory. Each of those fields has its own vocabulary, and the white paper necessarily draws on all of them. This guide makes every effort to use plain language wherever precision does not require otherwise, and to define technical terms when they first appear. A glossary is included in Appendix A for reference.

One term deserves clarification at the outset. When this document uses the word *minimalistic*, it does not mean incomplete in spirit. It means deliberately scoped to the smallest version that is still architecturally honest — a version that proves the core principles work, that connects all five systems in a real feedback loop, and that can be expanded without being rebuilt. Minimalistic is a discipline, not a compromise.

The white paper closes with an invitation: not for evangelism, but for serious work. This document is the beginning of that work — and the problems described above are the beginning of the serious work that needs to happen before, during, and long after the first line of code is written. Welcome to the build.

---

1. The Development Philosophy

1.1 Why We Build in Minimal Slices

There is a well-established failure mode in ambitious software projects: the system is designed in full, development begins on each component in parallel, and after months or years of work, the pieces are brought together for the first time — and they do not fit. Interfaces are incompatible. Assumptions made in isolation turn out to contradict each other. The system that emerges is not the system that was designed, and the gap between vision and reality is discovered too late to correct without enormous cost.

Integral is too important to build that way.

The alternative is to build in **vertical slices** — thin but complete paths through the entire system, from input to output, connecting all five subsystems in working form before expanding the capability of any one of them. The first slice will be narrow. A contributor will be able to propose something, design it, organize its production, log their contribution, and see the feedback loop return information to governance. That complete cycle — however simple — is the proof of concept. Everything else is elaboration.

This approach has a second advantage beyond risk reduction. It produces something real, early. Real software running on real data in a real cooperative context generates the kind of feedback that no amount of theoretical specification can replace. Comprehensive simulations and early nodes will surface edge cases, reveal underspecified interfaces, and expose assumptions that seemed reasonable in the abstract but fail under actual use. Building in slices means those discoveries happen while the system is still small enough to correct.

Depth in each system grows only after breadth across all five systems is established. This is the sequencing discipline that governs the entire project.

1.2 Architectural Faithfulness vs. Functional Completeness

The minimal slice approach carries a risk of its own: simplifying in ways that quietly violate the principles the system is built on, producing something that works as a prototype but cannot grow into the real architecture without being torn down and rebuilt.

This guide distinguishes carefully between two kinds of incompleteness.

Acceptable incompleteness means a module has not yet been built, a feature has been deliberately deferred to a later phase, or a field exists in a data structure but is not yet populated. The system is smaller than its full specification, but every part that exists is honest — it connects to the right interfaces, respects the right boundaries, and carries the right shape for what will eventually sit alongside it.

Unacceptable incompleteness means a foundational principle has been approximated away. The non-transferability of Integral Time Credits is violated to make accounting simpler. The feedback loop from FRS to CDS is stubbed out with a placeholder that never gets built. The OAD certification step is bypassed because it feels like overhead in early nodes. These are not simplifications — they are corruptions. A system built this way is not a minimal version of Integral; it is a different system with Integral's name.

The discipline this requires is specific: **before deferring anything, ask whether deferring it changes the nature of the system or merely reduces its scope**. If deferring a module means a principle goes unrepresented entirely — even in embryonic form — that module is not deferrable. If deferring a module means a capability is absent but the principle it serves is still honored by a simpler mechanism, the deferral is acceptable.

Section 3 applies this discipline to every module in every system, proposing what should be included in the minimum viable build, what should be deferred, and the reasoning behind each proposal. Those proposals are subject to ratification by the contributor community before development begins.

1.3 The Proof of Concept We Are Trying to Achieve

It is worth being precise about what the first working version of Integral needs to demonstrate — not in marketing terms, but in systems terms. The proof of concept succeeds if it shows, in working software operating in a real context, the following:

That a community can make a collective decision without a market or an authority. The CDS, even in minimal form, must produce a real decision that real people acted on — a decision that was structured, deliberated, recorded, and dispatched to the other systems. Not a vote, not a poll, but a governance signal that moved through the system.

That an open design can be community created, versioned, and used as the basis for production. The OAD, even in minimal form, must hold a certified design that COS actually uses to organize work. The design repository cannot be decorative — it must be the authoritative source of what gets built.

That labor contributions are recorded in a way that is transparent, non-transferable, and ecologically aware from the beginning. The ITC, even in minimal form, must log real hours against real tasks and produce an access cost that reflects embodied effort. Credits must not be transferable between participants, even in the prototype.

That production is organized from a design and generates the data that ITC and FRS need. The COS, even in minimal form, must take a certified OAD design, assign tasks, log completions, and produce a record that ITC can read to compute contribution values. The pipeline from design to production to contribution record must be a real pipeline, not a simulation.

That credits issued for labor are later extinguished when participants access produced goods. The ITC's access extinguishment mechanism — not just credit issuance — must be demonstrated in working form. Credits that are issued but never extinguished do not demonstrate the full contribution-to-access cycle that is central to Integral's non-monetary logic.

That operational data flows back into governance. The FRS, even in minimal form, must read something real from COS and ITC, produce a human-readable summary of what it finds, and submit that summary into CDS as a signal. The feedback loop — the cybernetic core of the entire system — must close. Even once, with simple data, in a constrained context. That closure is what distinguishes Integral from a collection of cooperative tools.

When all six of those things are true simultaneously the proof of concept is complete. Such a conclusion can be drawn from advanced modeling of the system and simulation and then tested in minimalistic real-world circumstances (proto-node).

---

2. The Five Systems at a Glance

2.1 What Each System Does (Plain Language)

Before examining how the five systems work together, it helps to understand what each one does on its own terms — not in technical detail, but in plain language that any contributor can hold in mind while reading the rest of this guide.

CDS — Collaborative Decision System

The CDS is how the Integral community node governs itself. It is the channel through which problems are raised — by participants directly, or surfaced as signals by other systems — proposals are deliberated, decisions are made, and those decisions are dispatched to the rest of the system for action. It is not a voting platform, though it can incorporate voting. It is not a discussion forum, though discussion happens within it. It is a structured governance pipeline — one that takes unorganized human concern and machine-generated system signals as input and produces legitimate, recorded, actionable decisions as output. Every significant action in Integral begins with a CDS decision and ends with CDS reviewing whether that action worked.

OAD — Open Access Design

The OAD is the system's shared knowledge base for how things are made — and more than a library, it is the information foundation that makes economic calculation possible. Before anything can be produced in Integral, a design for it must exist in the OAD: openly accessible, collaboratively refined, version-controlled, ecologically assessed, and certified as ready for production. Designs are contributed, tested against ecological and material constraints, optimized for efficiency and repairability, and published as certified blueprints that any node in the federation can use. But OAD's deeper function is to generate the computable design intelligence — labor-step decompositions, skill requirements, material and ecological coefficients, lifecycle and maintainability data — that COS depends on to coordinate production and that ITC depends on to calculate fair access values. Without that structured intelligence, neither production nor contribution accounting can be grounded in physical reality. Because designs are open and shared, improvements made by one node benefit every other node that uses the same design. Knowledge compounds rather than competes.

ITC — Integral Time Credits

The ITC is the system's contribution accounting mechanism. When a participant performs work that the community has identified as necessary — fabricating a component, driving a shuttle, maintaining equipment, documenting a process — that contribution is logged and recognized through time credits. Credits are not money and cannot be transferred, traded, or accumulated indefinitely. They decay over time to prevent stratification and hoarding. The ITC ledger records who contributed what, enabling proportional access to goods and services while maintaining a transparent record of collective effort. In this way, Integral preserves reciprocity between contribution and access without introducing the dynamics of a market economy.

Two boundaries are structural and non-negotiable. First, the ITC system applies only to work that is operationally necessary and node-mandated — meaning the node has identified the work as a genuine system need through its CDS process and linked it to COS as a recognized task. This criterion applies equally to physical and intellectual work. A contributor solving a critical structural problem in the ecological coefficient engine is doing operationally necessary work — if the node has mandated it — and generates ITCs just as a contributor fabricating a component does. What does not generate ITCs is routine democratic participation, standing governance involvement, and self-directed speculative ideation that has not been recognized as a system necessity. This is not a bias against intellectual work. It is a necessity-mandate criterion that prevents governance from being distorted by credit incentives and ensures that speculative or personal exploration does not draw on the commons in the way that recognized, necessary contribution does. The white paper is deliberate and coherent on this point: the exclusion is defined by mandate and necessity, not by the physical or intellectual character of the work. Second, the ITC system is designed to make itself progressively less necessary. As designs improve, automation increases, and cooperative production scales, access values drop proportionally. Entire categories of goods are expected to asymptotically approach zero contribution requirement, eventually exiting the ITC domain entirely. The system is not designed to institutionalize the accounting of labor indefinitely — it is designed to make abundant, low-burden access

to essential goods the normal condition of life in a mature node.

COS — Cooperative Organization System

The COS is where work actually happens. It takes certified designs from OAD and turns them into organized production — task lists, labor assignments, materials management, quality checks, and distribution. It is the operational layer of the system: the part that coordinates who does what, with what resources, by when. COS does not operate under direct production mandates from CDS. CDS sets policy, thresholds, and normative boundaries; within those boundaries, COS self-organizes around available designs, voluntary labor, and real material conditions. It is coordination without command.

COS also generates the raw operational data — hours worked, materials consumed, throughput rates, bottlenecks, quality outcomes — that the ITC uses to calculate contribution values and that the FRS uses to monitor system health. This data flow is not incidental. It is what makes economic calculation in Integral physically grounded rather than speculative. Without COS, the system has governance and designs but no production. Without COS's data, the system has no basis for fair contribution accounting or meaningful feedback.

FRS — Feedback & Review System

The FRS is the system's adaptive nervous system. It continuously monitors what is actually happening across COS, ITC, OAD, and CDS — tracking resource consumption, labor patterns, ecological indicators, design performance, access equity, production outcomes, and governance effectiveness — and converts that operational reality into signals the rest of the system can act on. But FRS does not only observe the present. It models the future: running counterfactual scenarios to identify viability boundaries, explore where constraints will bind, and surface risks before they become crises. When something is drifting out of alignment, FRS detects it early and surfaces it to CDS for deliberation. It does not make decisions or issue commands. It observes, diagnoses, models, recommends, and remembers. FRS findings and recommendations are advisory. A finding becomes actionable only when it is promoted to a CDS issue through the standard governance workflow, at which point it enters the deliberation pipeline with full democratic oversight. This boundary is structural, not procedural — it prevents FRS from becoming a form of algorithmic shadow governance, where the system's diagnostic authority quietly displaces the community's deliberative authority. FRS can detect and flag. It cannot act. Over time, as its memory deepens, FRS makes the entire system increasingly capable of learning from its own history.

FRS has a bidirectional relationship with OAD that is worth naming explicitly: OAD sends design events and ecological data into FRS as structured signals, and FRS sends operational feedback back into OAD — recalibrating ecological coefficients, revising lifecycle assumptions, and in cases where real-world performance diverges significantly from modeled expectations, recommending the review, revocation, or supersession of a certified design. That recommendation routes to OAD and ultimately to CDS for deliberation — FRS flags the divergence, but the decision belongs to the community. This makes FRS not just a monitor of production but an active contributor to the ongoing integrity of the design knowledge commons.

FRS also has a bidirectional relationship with CDS: FRS submits findings and recommendations to CDS, and CDS returns governance indicators — decisions dispatched, policies changed, issues resolved — so that FRS can assess whether governance responses to its findings are actually working. Without this return signal, the governance loop would close formally but never be evaluated.

2.2 How the Five Systems Depend on Each Other

Understanding each system individually is necessary but not sufficient. Integral's power — and its complexity — lies in how the five systems form a single integrated loop. No system is upstream of all the others. No system is purely downstream. Each one both provides and depends on the others in ways that make the whole qualitatively different from any of its parts.

The dependencies flow in a general cycle, but with cross-connections at every stage.

CDS initiates and closes the loop. A governance decision from CDS authorizes design work in OAD, production in COS, and monitoring parameters in FRS. At the other end, FRS feeds operational findings back into CDS, which reviews outcomes and issues revised or new decisions. CDS is both the starting point and the returning point of every significant system action.

OAD is the prerequisite for COS, and a bidirectional partner to both ITC and FRS. Nothing enters production without a certified design. COS cannot organize work around an uncertified or nonexistent blueprint. This dependency is architectural, not procedural — it must be enforced in the software, not just in policy. OAD also receives feedback from multiple directions: from COS, when production reveals problems with a design; from ITC, when contribution and access data indicate a design is too labor- or material-intensive; and from FRS, when operational monitoring reveals that a design's real-world ecological coefficients, labor intensity, or lifecycle assumptions diverge from what was modeled at certification. When ITC or FRS data indicate a design is too costly in labor or materials, OAD triggers a review that prompts the responsible design teams to explore revisions. In cases where real-world performance diverges significantly from modeled expectations, FRS can recommend revocation or supersession of a certified design — but that recommendation routes to OAD, and the decision belongs to OAD and ultimately to CDS. FRS flags the divergence; it does not pull the trigger. OAD is not a static repository. It is a living knowledge commons that the whole system keeps honest.

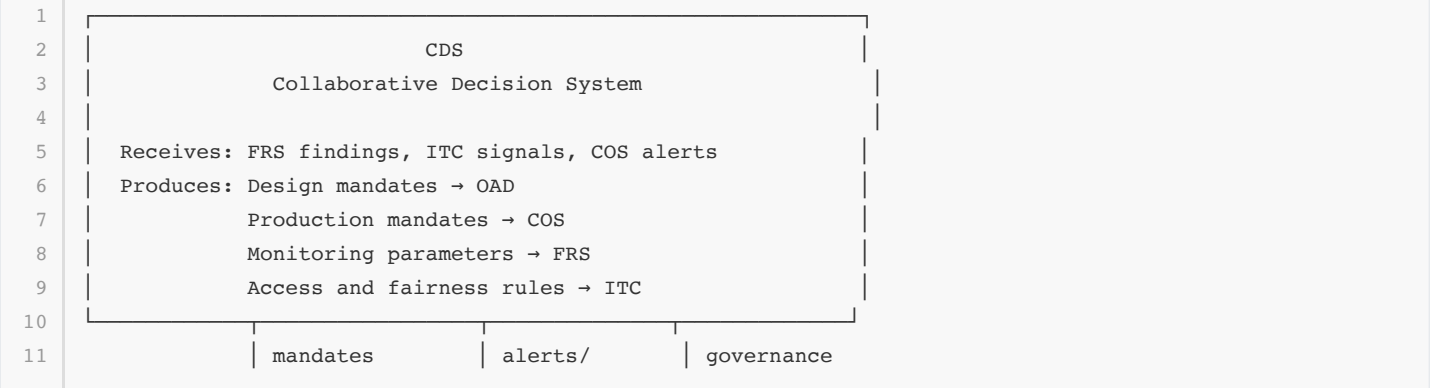
COS is the source of truth for ITC and FRS. The labor events, material consumption records, and production outcomes that COS generates are the raw inputs that ITC uses to compute contribution values and access costs, and that FRS uses to assess system health. If COS data is incomplete or unreliable, both ITC and FRS are compromised. This makes COS data integrity a foundational concern of the entire system.

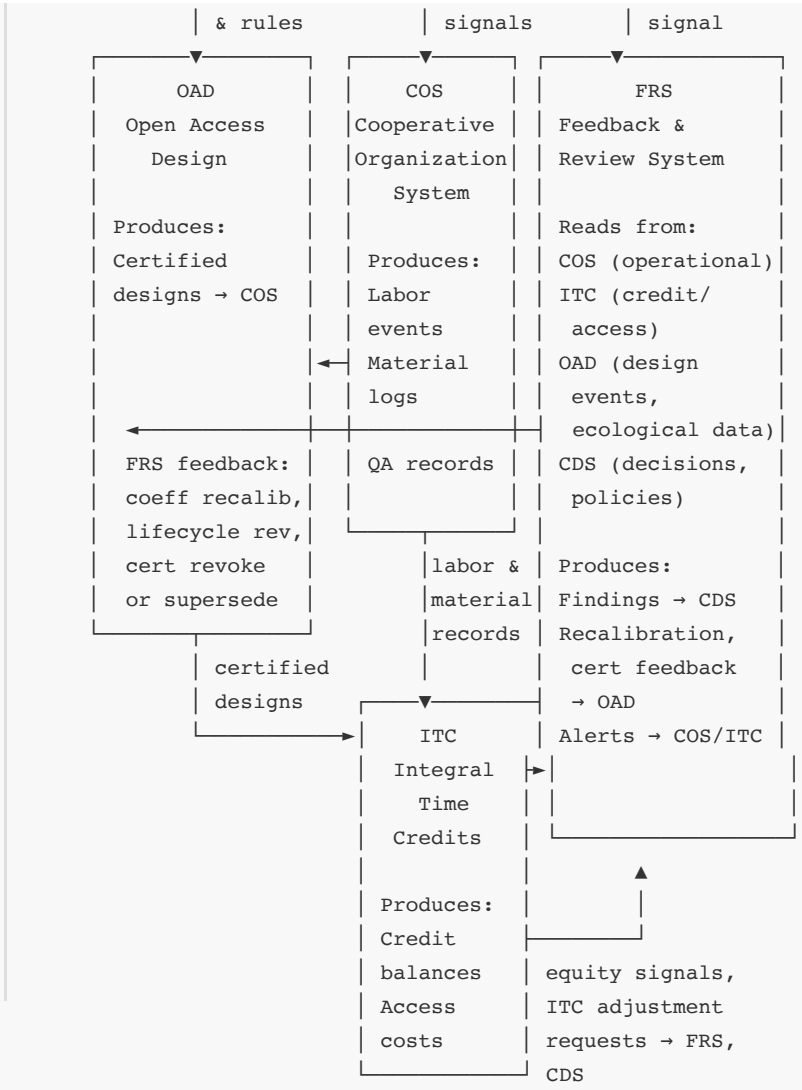
ITC depends on COS, generates the ledger that FRS reads, and signals back to OAD and COS. ITC reads from COS to calculate credits and access costs, and maintains a transparent append-only record of all contribution, decay, redemption, and access events. When ITC data indicates that certain goods require redesign or that production capacity needs adjustment, those signals inform OAD and COS directly. ITC's Module 7 detects specific ethical violations — coercion, preferential treatment for high-balance holders, attempts to corner specialized roles — and escalates them through FRS to CDS for democratic resolution. The broader diagnostic work — identifying systemic patterns of inequity, access strain, or contribution imbalance across the whole system — belongs to FRS, which reads ITC's ledger as one of its primary signal sources. ITC generates the data, flags violations, and signals directly when redesign or capacity adjustments are needed. FRS finds the deeper patterns.

FRS depends on all four other systems and reports back to all of them. FRS reads from COS (operational data), ITC (contribution and access patterns), OAD (design outcomes and ecological assessments), and CDS (governance decisions and policy changes). It synthesizes these into advisory diagnostic findings and routes non-executive recommendations to the appropriate system — design reviews to OAD, workflow adjustments to COS, valuation and fairness signals to ITC, and governance prompts to CDS. Recommendations do not trigger automatic changes in any system. They become actionable only when the receiving system acts on them through its own processes, or — in the case of CDS-targeted recommendations — when they are formally accepted as governance issues and enter the deliberation pipeline. Reading from CDS as well as reporting to it closes the governance evaluation loop: FRS can now assess not just what the system produces, but whether the governance process responding to its findings is working. FRS is the integrating layer that makes the system aware of itself.

2.3 The Master Data Flow Map

The following diagram describes, in simplified form, how data moves between the five systems in normal operation. This is not an exhaustive technical specification — it is an orientation map. Detailed interface specifications appear in Section 4.





The OAD↔FRS relationship is bidirectional and deserves explicit attention. OAD sends design events, certification changes, and ecological assessment data into FRS as structured inputs. FRS sends operational feedback back into OAD — recalibrating ecological coefficients when real-world material degradation or scarcity diverges from modeled values, revising lifecycle assumptions when deployed goods perform differently than certified. When FRS feedback shows significant divergence between modeled and real-world performance, certification can be revoked or the design superseded by a revised version. This feedback loop is what prevents the OAD knowledge commons from becoming a static archive of assumptions that no longer reflect reality.

The CDS↔FRS relationship is also bidirectional — a detail the diagram makes explicit. FRS reports findings to CDS to close the operational feedback loop. CDS returns governance indicators to FRS — decisions dispatched, policies changed, issues opened and resolved — so that FRS can assess whether governance decisions are actually producing their intended effects. Without this return signal, FRS would know what the system produces but not whether the governance process responding to its findings is working.

The diagram also shows that ITC and COS both send signals directly to CDS — not only through FRS. ITC adjustment requests and COS capacity flags enter CDS as direct inputs alongside FRS findings, giving the governance system a complete picture of operational reality rather than a filtered one. FRS synthesizes and diagnoses; it does not monopolize the signal path to governance.

The map shows that every system both gives and receives. There are no purely passive consumers of data in this architecture. This mutual dependency is what makes the system coherent as a whole — and what makes the integrity of each system's data output a responsibility to every other system that depends on it.

2.4 What "Minimalistic but Faithful" Means in Practice

It is worth making sure this phrase is concrete before moving into the module-level detail of Section 3.

Consider the ITC's time-decay mechanism. In the full white paper specification, decay is a nuanced function that accounts for contribution type, node context, and federated calibration. In the minimal first implementation, decay uses an exponential half-life model with an inactivity grace window — exactly as specified in the white paper — governed by a CDS-approved DecayRule record. What is simpler in Phase 2 is not the mechanism but the policy parameters: the CDS-approved rule starts with a conservative half-life and grace window rather than sophisticated per-contribution-type calibration. The *principle* — that credits cannot be accumulated indefinitely, that the system structurally prevents the emergence of a class of permanent credit-holders — is fully present from day one. The data structure that holds a credit balance already has fields for half-life and grace window. When more sophisticated decay policy is introduced in a later phase, it replaces the referenced rule parameters. It does not require redesigning the schema.

Contrast this with a decision to skip the OAD certification step in early nodes because it feels like unnecessary overhead. A design goes directly from proposal to production without passing through a certification gate. This is not a simplification — it is a violation. The principle that production requires certified design is not a feature of OAD; it is a foundational constraint of the entire system. Bypassing it, even temporarily, means COS has no authoritative source of truth for what it is building, ITC has no reliable basis for ecological flagging, and FRS has no design baseline against which to assess production outcomes. The system is structurally broken even if it appears to function locally.

The test, in every case, is this: **if this simplification were never revisited, would the system still be recognizably Integral?** If the answer is yes, the simplification is acceptable. If the answer is no, the principle it violates must be represented — however simply — from the beginning.

Section 3 applies this test to every module in every system. The decisions recorded there are not final — they will be refined as the contributor community engages with them — but they represent a serious first attempt to draw the line between acceptable scope reduction and unacceptable principle violation.

---

3. The Minimum Viable Module Set

A Proposal, Not a Decree

Everything in this section is a proposal.

The module selections, deferral decisions, and simplification choices described here represent one attempt to answer a hard question: what is the smallest version of each system that is still architecturally honest? That question has no single correct answer. The answer depends on what the community values, what its first real operational context demands, and what risks it is willing to accept in exchange for earlier working software.

Contributors are explicitly invited to challenge any deferral, propose alternatives, and open GitHub issues against specific decisions in this section. The issue templates in the `integral/decisions` repository include a template specifically for architectural challenges.

3.1 CDS — Collaborative Decision System (Proposed Core Modules)

The full CDS specification in the white paper describes ten modules, moving from initial signal intake through structured deliberation, constraint checking, weighted consensus, transparent versioning, implementation dispatch, human resolution, and post-decision review. In the minimal first implementation, the CDS does not need all of that machinery. It needs to demonstrate one thing clearly: **that a community can raise an issue, deliberate around it, produce a legitimate decision, and dispatch that decision to the rest of the system — with a complete, tamper-evident record of the process.**

NOTE: An honest warning about human mediation. Several modules below are marked as "manually assisted," meaning that a human facilitator performs the work that software will eventually do. This is architecturally acceptable but practically fragile, and the window in which it remains sustainable is shorter than it might appear.

In the early stage — with a handful of contributors making a small number of decisions — one person can hold the facilitation role together through attention and goodwill. In later stages — with multiple working groups, decisions arriving from five system domains simultaneously, and a growing contributor base — that person becomes a bottleneck. The mechanical work of formatting issues, timing comment periods, sending reminders, and checking whether submissions meet the required fields accumulates rapidly. Under that load, facilitation quality degrades, decisions back up, and the community begins routing around governance informally — exactly the failure mode the CDS is designed to prevent.

The software should handle all mechanical work from the earliest possible point, leaving only judgment to humans. Formatting a proposal, opening a comment period, sending a reminder, and logging a decision record — these are mechanical. Determining whether a blocking objection is substantive, synthesizing a genuine consensus from contested positions, and writing a decision record that captures the reasoning honestly — these require judgment and human attention.

For this reason, basic CDS facilitation tooling is treated as an *early, primary deliverable*, not a later-stage automation project. This does not mean building the full algorithmic consensus engine — that genuinely requires operational history and can wait. It means building the scaffolding that makes human facilitation sustainable: structured submission forms that enforce required fields, automated comment-period timers, bot-driven reminders 48 hours before close, and a facilitator checklist interface that guides synthesis without leaving it to memory.

IMPORTANT: Creating the CDS as a core initial focus, and giving it priority over other Integral systems development, allows it to be used for the development process itself as the developer community grows more complex.

The following modules are included in the minimum viable CDS.

Module 1 — Issue Capture & Signal Intake *(Included)*

This module is included in full principle, though simplified in implementation. In the minimal version, issue capture accepts human-submitted proposals and concerns through a simple authenticated form interface. Automated signals from FRS, ITC, and COS — which in the full system feed into Module 1 continuously — are supported in the data schema from day one, but initially they arrive as manually submitted summaries rather than live system feeds. Every submission is timestamped, identity-tagged, and stored as a structured issue object. The authentication mechanism can be simple — a verified participant account within the node — but the concept of authenticated, non-duplicated input must be enforced from the beginning.

What is deferred: Automated signal ingestion from other systems. Real-time deduplication and normalization at scale.

Module 2 — Issue Structuring & Framing *(Included, manually assisted)*

In the full specification, Module 2 uses semantic clustering and argument mapping to automatically organize raw submissions into coherent problem frames. In the minimal version, this function is performed by a human facilitator working within a structured template — a form that prompts for scope, related submissions, implicit assumptions, and what the decision actually is. The output is the same structured issue view that downstream modules consume. The data structure is identical to what the automated version will eventually produce. The human does the cognitive work that the algorithm will later do.

What is deferred: Automated semantic clustering. AI-assisted argument mapping. Dependency graph generation.

Module 6 — Weighted Consensus Mechanism *(Included, simplified)*

In the full specification, Module 6 is a mathematically sophisticated engine that evaluates preference gradients, identifies blocking objections, and produces outcomes across a spectrum from full approval to conditional approval to revision requests. In the minimal version, consensus is recorded through a structured template that captures: level of support (strong support / support / neutral / concerns / strong objection), the reasoning behind each position, and any conditions attached to approvals. A human facilitator synthesizes these into a decision outcome using the same categories the full engine will eventually produce algorithmically. The decision record uses the same data schema the automated engine will eventually populate.

What is deferred: Algorithmic preference synthesis. Automated blocking objection detection. Quadratic or weighted voting mechanisms.

Module 7 — Decision Recording, Versioning & Accountability *(Included in full principle)*

This module is non-negotiable and is included from day one without simplification of principle. Every issue, every submission, every deliberation record, and every decision outcome is stored in an append-only record with a timestamp and a link to the participant or system that produced it. In the minimal version, this is implemented as a structured database with append-only write rules rather than a cryptographically chained ledger — but the transition to cryptographic chaining later must not require schema changes. The record structure anticipates that upgrade from the beginning. No decision in Integral exists without a traceable record of how it was reached. This is true from the first day of operation.

What is deferred: Cryptographic hash chaining. Public dashboard interface. Federated record synchronization.

Module 8 — Implementation Dispatch Interface *(Included, simplified)*

Once a decision is recorded, it must go somewhere. In the minimal version, dispatch is a structured output document — a decision packet — that specifies what action is required, which system is responsible, what parameters apply, and what FRS should monitor as a success indicator. Initially, this packet may be acted on manually by the relevant system operators rather than consumed by an automated API. But the packet format is standardized from the beginning, because it is the primary contract between CDS and every other system.

What is deferred: Automated API dispatch. Event-driven system integration. Real-time orchestration.

Modules Deliberately Deferred

Modules 3, 4, and 5 are deferred from the minimal build. Module 2 is included in manually assisted form, as described above.

Module 3 — Knowledge Integration & Context Engine is the module that prevents the governance process from making the same mistake twice. It brings relevant existing knowledge — prior decisions, design outcomes, FRS history, ecological data — into the deliberation space so that each new proposal is considered with full institutional context rather than in isolation. Initially, a designated reviewer performs this function manually: searching the decision record, the FRS archive, and the specifications repository before deliberation opens. The automated engine will eventually do this continuously and comprehensively. The manual version establishes the discipline.

Module 9 (Human Deliberation & High-Bandwidth Resolution) is partially present by default, since early CDS operation will be heavily human-facilitated anyway. However, its formal role in the full system warrants explicit description. CDS Module 9 is the constitutional last resort — invoked when computational consensus cannot resolve a dispute because the issue involves cultural meaning, ethical tension, or irreducible value conflict that algorithmic inference cannot navigate. When invoked, Module 9 escalates to **Syntegrity**: a high-bandwidth human deliberation architecture developed by cybernetician Stafford Beer. Unlike traditional debate or parliamentary procedure, Syntegrity is a structured communication protocol that distributes influence evenly, routes insight through designed rotation, and surfaces coherence that cannot be derived from argument trees or scoring algorithms. Syntegrity does not replace CDS — it is a specialized mechanism inside Module 9 for resolving complexity when conventional deliberation is insufficient. Outputs from a Syntegrity session re-enter the formal CDS pipeline through Module 7 recording and Module 8 dispatch, preserving continuity and legitimacy. In the minimal version, Module 9 operates through facilitated human

deliberation without Syntegrity's formal protocol; the data structures for recording and dispatching its outputs are present from day one.

Module 10 (Review, Revision & Override) is deferred but its trigger conditions are recorded in every decision packet from day one, so that when Module 10 is built it has a complete history of decisions to operate on.

3.2 OAD — Open Access Design (Proposed Core Modules)

The full OAD specification in the white paper describes ten modules covering collaborative design, material and ecological assessment, lifecycle modeling, feasibility simulation, labor decomposition, systems integration, optimization, validation, and a knowledge commons repository. In the minimal first implementation, OAD needs to demonstrate one thing: **that a design can be created, ecologically flagged, certified as ready for production, and used by COS as an authoritative source of what is being built.**

Module 1 — Design Submission & Structured Specification *(Included, simplified)*

The minimal OAD begins with a design submission interface — a structured form that captures the design's name, version, description, bill of materials, intended production steps, and an initial ecological impact flag (low / medium / high). Initially, designs are authored by individuals or small groups and submitted for review rather than being co-authored in real time through a collaborative workspace. Version control is present from day one — every submitted design is a versioned object, and revisions create new versions rather than overwriting prior ones. This is non-negotiable because the white paper's entire OAD logic depends on design lineage being traceable.

What is deferred: Real-time collaborative editing. Automated version diffing. Dependency graph tracking between designs.

Module 2 — Collaborative Design Workspace *(Partially included)*

Module 2 is partially present through the structured submission form introduced in Module 1, which supports individual and small-group authorship. Real-time multi-user collaborative editing is not yet implemented. The submission form establishes the data structures that a full collaborative workspace will eventually populate without schema changes.

What is deferred: Real-time collaborative editing. Multi-user co-authorship workflows. Live conflict resolution.

Module 3 — Material & Ecological Coefficient Engine *(Included, minimally)*

This module is included in minimal form because its principle — that ecological impact is assessed before production, not after — is foundational to the system's integrity. In the minimal version, ecological assessment is a human-completed checklist attached to every design submission. It asks: what materials are required, are any flagged as ecologically sensitive, what is the estimated end-of-life pathway, and what is the overall impact rating. The checklist is simple. The data fields it populates are not — they are the same fields that the full Material and Ecological Coefficient Engine will eventually compute automatically. A design cannot be certified without a completed ecological assessment, even in the minimal build.

What is deferred: Automated lifecycle assessment. Standardized ecological coefficient databases. Material substitution recommendation engines.

Module 9 — Validation, Certification & Release *(Included, simplified)*

Certification is the gate between design and production, and it must exist in the minimal build. In the minimal version, certification is a structured review process performed by a designated group of node participants — not a single authority, but a small, accountable review panel whose membership and decisions are recorded in the OAD ledger. A design is certified when it has passed ecological assessment, has a complete and coherent bill of materials, and has been reviewed and approved by the certification panel. Certified designs are marked with a version-locked status that COS can verify before beginning production. Uncertified designs cannot enter production. This constraint is enforced architecturally.

What is deferred: Automated safety and performance simulation. Cross-node certification recognition. Federated certification standards.

Module 10 — Knowledge Commons & Reuse Repository *(Included, as the primary interface)*

In one sense, the entire minimal OAD is a knowledge commons — a repository of certified designs that any node participant can browse, reference, and build upon. In the minimal version, the commons is a structured database with search and version history. It is included because without it, OAD is not a shared resource — it is just a filing system. The commons framing, even in minimal form, establishes the principle that designs belong to the network, not to their authors.

What is deferred: Federated repository synchronization across nodes. Automated reuse recommendation. Design impact analytics.

Modules Deliberately Deferred

Modules 4 (Lifecycle & Maintainability Modeling), 5 (Feasibility & Constraint Simulation), 6 (Skill & Labor-Step Decomposition), 7 (Systems Integration & Architectural Coordination), and 8 (Optimization & Efficiency Engine) are all deferred from the minimal build. Their functions will be approximated by the human judgment of the design authors and certification panel in early nodes.

3.3 ITC — Integral Time Credits (Proposed Core Modules)

The full ITC specification describes nine modules covering labor event capture, skill and context weighting, time decay, labor-budget forecasting, access allocation, cross-cooperative reciprocity, fairness safeguards, ledger transparency, and system integration. In the minimal first implementation, ITC needs to demonstrate one thing: **that labor contributions are recorded honestly, that credits reflect embodied effort and carry a simple ecological dimension, that they decay over time, and that they cannot be transferred between participants.**

These four properties — honest recording, embodied valuation, decay, and non-transferability — are the ITC's foundational principles. All four must be present and architecturally enforced from day one. None of them are deferrable.

Module 1 — Labor Event Capture & Verification *(Included)*

Every unit of recognized labor in the system produces a labor event record. In the minimal version, this record contains: participant identity, task reference (linked to a COS task), hours worked, skill tier, and a timestamp. Verification is peer-based — a second participant or a COS task supervisor confirms the event. The confirmation is logged alongside the original record. Unverified labor events do not generate credits. This constraint is enforced from day one because unverified contribution claims would immediately undermine the system's integrity.

What is deferred: Biometric or cryptographic self-attestation. Automated verification through COS workflow completion signals. Dispute resolution workflows.

Module 2 — Skill & Context Weighting Engine *(Included, simplified)*

In the full specification, skill weighting is a nuanced function accounting for relative scarcity of skills, contextual urgency, and node-level calibration. In the minimal version, weighting is implemented as a simple tiered multiplier: a small number of skill tiers — low, medium, high, and expert — each carrying a defined weight band. The weights are set by CDS and recorded as node policy. The weighting engine is simply a lookup against this CDS-approved table. The data structure that holds a credit calculation already carries a field for the weighting method used, so that more sophisticated weighting functions can be substituted later without schema changes.

The weighting engine must accept 1.0 as a valid multiplier for all tiers by default. This ensures the system can operate on a pure one-to-one basis — one hour of any skill tier equals one credit — without requiring schema changes if the community determines that flat recognition is the right policy. The `expert` tier in particular should be understood as a scarcity flag for FRS monitoring purposes, not as a declaration that expert labor is intrinsically more valuable. Whether differential weighting is warranted is a governance question that CDS resolves through policy, not an architectural assumption the system makes in advance.

What is deferred: Dynamic skill scarcity detection. Contextual urgency modifiers. Federated weighting calibration.

Module 3 — Time-Decay Mechanism *(Included, simplified)*

Non-accumulation is a foundational principle of the ITC. It must be present from the first credit ever issued. In the minimal version, decay is implemented as an **exponential half-life function with an inactivity grace window** — consistent with the white paper specification — applied on a regular automated schedule and governed by a CDS-approved DecayRule record. The decay mechanism is not simplified in Phase 2; what is simplified is the CDS policy that governs it. Phase 2 begins with a conservative, straightforward half-life value and grace window set by CDS, rather than sophisticated per-contribution-type or federated calibration. The schema includes `decay_half_life_days` and `decay_inactivity_grace_days` from day one. Decay affects unused balances only — credits extinguished through access are not subject to decay — and decay is not a punishment for inactivity but a normalization mechanism that keeps access aligned with present participation. The decay rule must reference a **CDS-approved DecayRule record** rather than a hardcoded value. Later, more sophisticated rules — contribution-type-sensitive rates, need-adjusted exemptions, federated calibration — replace the policy parameters without requiring schema changes. The governance architecture for CDS to own decay policy is present from the beginning, not retrofitted later.

What is deferred: Contribution-type-sensitive decay rates. Need-adjusted decay exemptions. Federated decay calibration across nodes.

Module 5 — Access Allocation & Extinguishment *(Included)*

Credits must be extinguishable against goods and services, or they mean nothing. In the minimal version, access allocation works as follows: every certified OAD design carries an ITC access cost computed from its labor and material inputs and its ecological flag. When a participant wishes to access a produced good, the system checks their credit balance against the access cost and, if sufficient, **extinguishes** the corresponding credits and records the event. Credits are not transferred to another participant or entity — they cease to exist. This extinguishment is what prevents accumulation, trade, speculation, and the conversion of contribution into lasting power. Need-based adjustments — where participants with reduced capacity receive modified access requirements — are included from the beginning as a simple override field on the access cost, set by CDS policy for defined categories of need. The principle that access is never purely transactional must be present from the start.

Note: Extinguishment is distinct from Time-Decay. Extinguishment is triggered by a participant permanently acquiring a produced good — credits are actively destroyed at the moment of access. Time-Decay (Module 3) is an automated, scheduled reduction of unused balances over time. Both prevent accumulation, but through different mechanisms for different purposes. Contributors should never conflate the two.

Note on borrowing: Short-term borrowing of shared-use items — tools, equipment, infrastructure — does not extinguish credits. Only permanent acquisition does. This distinction is present in the data schema from the beginning as an `access_type` field on every access event: `permanent_acquisition` (credits extinguished) or `shared_use` (credits not extinguished).

What is deferred: Dynamic access cost recalculation as production efficiency improves. Automated need-assessment workflows. Cross-node access recognition. Access Centers, Tool Libraries, and Distribution Nodes as dedicated physical infrastructure (present conceptually from the beginning; formalized later as named infrastructure).

Module 8 — Ledger, Transparency & Auditability *(Included in full principle)*

Like CDS Module 7, the ITC ledger is non-negotiable. Every credit issued, every decay event, every extinguishment, and every balance adjustment is recorded in an append-only ledger with a timestamp and a reference to the event that triggered it. In the minimal version, this is a structured append-only database. The transition to a cryptographically chained ledger later requires no schema changes. Every participant can view their own full credit history, and node administrators can produce aggregate reports for CDS review. The ledger is never private and never editable.

What is deferred: Public-facing transparency dashboard. Cryptographic ledger chaining. Federated ledger reconciliation.

Modules Deliberately Deferred

Module 4 (Labor-Budget Forecasting & Need Anticipation), Module 6 (Cross-Cooperative & Internodal Reciprocity), Module 7 (Fairness, Anti-Coercion & Ethical Safeguards), and Module 9 (Integration & Coordination) are deferred from the minimal build. Modules 6 and 9 are federation-dependent and cannot be meaningfully implemented before multiple nodes exist. Module 7's fairness monitoring function will be approximated by FRS signal review and CDS oversight. Module 4 requires operational data history that does not yet exist in a new node.

3.4 COS — Cooperative Organization System (Proposed Core Modules)

The full COS specification describes nine modules covering production planning, labor organization, resource procurement, workflow execution, capacity and constraint balancing, distribution, quality assurance, inter-cooperative coordination, and ledger transparency. In the minimal first implementation, COS needs to demonstrate one thing: **that a certified design can be translated into organized work, that labor and materials are tracked as that work proceeds, and that the resulting records are reliable enough for ITC and FRS to use.**

COS is, in a real sense, the engine room of the entire system. Its data quality determines the integrity of everything downstream.

Module 1 — Production Planning & Work Breakdown *(Included, simplified)*

When CDS authorizes production of a certified OAD design, COS Module 1 creates a production plan — a structured breakdown of the work into tasks. In the minimal version, this breakdown is created manually by a node coordinator using a standard task template. Each task carries: a reference to the OAD design and version, a description of the work, the skill tier required, an estimated labor hours figure, any material inputs required, and a status field. The production plan as a whole carries a reference to the CDS decision that authorized it. Nothing enters production without both a certified design reference and an authorizing CDS decision reference. Both constraints are enforced architecturally.

What is deferred: Automated task decomposition from OAD design specifications. Constraint-aware scheduling. Capacity-based workload balancing.

Module 2 — Labor Organization & Skill-Matching *(Included, simplified)*

In the minimal version, labor assignment is a combination of self-selection and coordinator facilitation. Participants browse available tasks, indicate their skill tier and availability, and are assigned to tasks by a coordinator who confirms the match. Every assignment is recorded with the participant's identity, the task reference, the assigned hours, and the timestamp of assignment. Completion is recorded separately when the task is done and peer-verified. The assignment and completion records are the

primary inputs to ITC Module 1.

What is deferred: Automated skill-matching algorithms. Availability forecasting. Dynamic reallocation under capacity constraints.

Module 3 — Resource Procurement & Materials Management *(Included, simplified)*

Every task that consumes materials generates a materials record: what material, what quantity, what the ecological impact flag is (inherited from the OAD design's ecological assessment), and what remains in inventory after consumption. In the minimal version, inventory is tracked manually against a node materials ledger. Procurement from outside the node is recorded as an external dependency event — a flag that FRS monitors. The materials record is a structured data object from day one, because ITC uses it to compute the material component of access costs and FRS uses it to assess ecological and supply chain health.

What is deferred: Automated inventory management. Cross-node materials matching. Procurement optimization.

Module 4 — Cooperative Workflow Execution *(Included)*

This is the operational core of the minimal COS — the ongoing management of work in progress. Task status moves through a defined lifecycle: planned, assigned, in progress, pending verification, verified, complete. Every status transition is recorded with a timestamp and the identity of the participant or coordinator who triggered it. The workflow is simple but disciplined. No task can move from in-progress to complete without a verification step. No material can be logged as consumed without a corresponding task reference. These constraints are enforced in the data model, not just in policy.

What is deferred: Parallel workflow management across large production teams. Automated bottleneck detection. Real-time workflow visualization.

Module 7 — Quality Assurance & Safety Verification *(Included, simplified)*

Quality assurance in the minimal version is a structured review step that occurs before any produced good is logged as complete and before distribution occurs. The review captures: whether the output matches the certified OAD design, whether any deviations occurred, and whether the output meets the safety and quality standards specified in the design. Failed QA is recorded — not discarded — because failure data is valuable to FRS and feeds back into OAD for design revision. A produced item that fails QA does not generate ITC credits for distribution. QA records are linked to the OAD design version, the COS production plan, and the ITC valuation record.

What is deferred: Standardized testing protocols. Automated quality metrics. Cross-node quality benchmarking.

Module 9 — Transparency, Ledger & Audit *(Included in full principle)*

As with CDS Module 7 and ITC Module 8, the COS ledger is non-negotiable from day one. Every production plan, task assignment, labor event, material consumption, QA record, and distribution event is recorded in an append-only ledger. The ledger is the source of truth for ITC valuation and FRS monitoring. In the minimal version it is a structured append-only database. No record is ever deleted or modified — only appended to with correction events that reference the original.

What is deferred: Cryptographic ledger chaining. Federated ledger sharing. Real-time public audit dashboards.

Modules Deliberately Deferred

Module 5 (Capacity, Throughput & Constraint Balancing), Module 6 (Distribution & Access Flow Coordination), and Module 8 (Cooperative Coordination & Inter-Coop Integration) are deferred from the minimal build. Distribution is handled as a simple recorded handoff event within Module 4's workflow. Inter-cooperative coordination requires multiple nodes and is federation-dependent.

3.5 FRS — Feedback & Review System (Proposed Core Modules)

The full FRS specification describes seven modules covering signal collection, diagnostic analysis, constraint modeling and scenario simulation, recommendation routing, democratic sensemaking, longitudinal memory, and federated intelligence exchange. In the minimal first implementation, FRS needs to demonstrate one thing: **that operational data from COS, ITC, and OAD is read, interpreted, and returned to the appropriate systems as structured signals — including closing the feedback loop to CDS that is the cybernetic core of the entire system.**

FRS reads from four source systems: COS (labor events, material consumption, QA outcomes, throughput data), ITC (credit flows, access patterns, equity distribution), OAD (design certification events, ecological assessments, design version changes), and CDS (governance decisions dispatched, policies updated, issues resolved). It routes recommendations to four target systems: CDS (governance findings), OAD (ecological coefficient recalibration, lifecycle revision, certification revocation triggers), COS (workflow and capacity adjustments), and ITC (valuation and fairness signals).

Even if FRS in the minimal version does nothing more than produce a weekly summary that a human presents to CDS as an agenda item, that loop must close. A system without feedback is not Integral — it is a collection of cooperative tools.

Module 1 — Signal Intake & Semantic Integration *(Included, simplified)*

In the minimal version, FRS Module 1 reads from the COS, ITC, OAD, and CDS ledgers and event records on a scheduled basis — weekly is appropriate for early nodes — and produces a structured signal packet summarizing what occurred during the period. From COS the packet draws: total labor events logged, total materials consumed by category and ecological flag, any QA failures, and any tasks that exceeded estimated hours significantly. From ITC it draws: total credits issued and decayed, total access extinguishments, and balance distribution across participants. From OAD it draws: any new certifications issued, any design versions updated, and any ecological assessment changes. From CDS it draws: decisions dispatched, policies updated, governance issues opened and closed, and decision records merged — enabling FRS to assess whether governance decisions are producing their intended effects. **This packet is generated by automated aggregation scripts running scheduled queries against all four system databases — not manually compiled.** The packet format is standardized from day one because it is the input to every downstream FRS module.

What is deferred: Real-time continuous signal ingestion. Automated anomaly detection during data collection. Multi-system event streaming.

Module 2 — Diagnostic Classification & Pathology Detection *(Included, simplified)*

In the minimal version, diagnostic analysis is a human-assisted process. A designated FRS reviewer reads the signal packet produced by Module 1 and works through a structured diagnostic checklist: Are any ecological flags trending upward? Are labor hours significantly over or under estimates? Are any materials showing supply strain? Are credit balances distributing equitably across participants? Are there QA failure patterns that suggest design problems? The checklist produces a set of named findings — structured objects with a finding type, a severity level, and a reference to the evidence in the signal packet. These structured findings, even when produced by a human, use the same data schema that automated diagnostic algorithms will eventually populate.

What is deferred: Automated pattern detection and anomaly classification. Persistence tracking across multiple signal periods. Statistical drift analysis.

Module 4 — Recommendation & Signal Routing *(Included, simplified)*

For each finding produced by Module 2, Module 4 produces a recommendation — a structured, non-executive suggestion routed to the appropriate system. In the minimal version, recommendations are human-authored using a standard template that specifies: the target system (CDS, OAD, COS, or ITC), the recommendation type, the finding it responds to, and the suggested action. Recommendations are never commands. They are inputs to the systems they address, which retain full authority over whether and how to act on them. A CDS-targeted recommendation automatically becomes a candidate agenda item for the next governance cycle, but it does not enter the deliberation pipeline until it is formally accepted as a governance issue — the community retains full authority over whether and how to act on it. This boundary is what prevents FRS diagnostic authority from quietly displacing democratic deliberative authority.

What is deferred: Automated recommendation generation from diagnostic findings. Severity-based routing priority. Federated recommendation sharing.

Module 5 — Democratic Sensemaking Interface *(Included, as the primary FRS output)*

The sensemaking interface is what makes FRS useful to the community rather than just to system administrators. In the minimal version, it is a structured summary document — produced from the signal packet and findings — that presents the state of the node in plain language: what was produced, what was contributed, what resources were consumed, what concerns were detected, and what recommendations have been made. This document is published to all node participants and submitted to CDS as a standing agenda item. It is the community's view of its own operational reality.

What is deferred: Automated real-time dashboards. Interactive data visualization. Participant-facing query interfaces.

Modules Deliberately Deferred

Module 3 (Constraint Modeling & Scenario Simulation), Module 6 (Longitudinal Memory, Pattern Learning & Institutional Recall), and Module 7 (Federated Intelligence & Inter-Node Learning) are deferred from the minimal build. Module 3 requires sufficient operational history to model meaningfully — it will become valuable once a node has several months of data. Module 6 similarly depends on accumulated history but its data begins accumulating from day one. Module 7 is federation-dependent and cannot be meaningfully implemented before multiple nodes exist.

3.6 What Is Proposed for Deferral — and Why

Almost everything proposed for deferral from the minimum viable build falls into one of four categories.

Automation of functions that humans can perform — with a critical constraint. Semantic clustering, automated diagnostic analysis, algorithmic consensus synthesis, and weighted recommendation routing are all cases where a human working from a structured template can perform the same function at small scale. The data schemas are present. The human is the algorithm, temporarily. This is acceptable because it does not compromise the principle — it only reduces throughput.

However: humans are not reliable algorithms and should not be asked to act like them for long. The distinction that must be enforced from the earliest build of each system is between **mechanical work** (data aggregation, record compilation, format conversion) and **judgment work** (diagnosis, interpretation, recommendation). Mechanical work must be automated from the moment the system is built — the software compiles the signal packet, produces the structured template, runs the aggregation queries. Judgment work is what the human contributes. An FRS operator who is asked to manually compile raw database records into a weekly report will burn out within months and the feedback loop will break when they do. An FRS operator who reviews a software-generated summary and applies their judgment to what it means is doing sustainable, valuable work. This applies to CDS facilitation tooling and to FRS signal aggregation equally — in each case, the automated scaffolding is a non-negotiable deliverable of the early build, not something to be added later.

Federation-dependent capabilities. Cross-node reciprocity, federated ledger synchronization, internodal material matching, and federated intelligence exchange all require multiple nodes to exist before they can be meaningfully implemented. Building them before the network exists is premature and likely to produce interfaces that don't survive contact with real federation conditions.

History-dependent capabilities. Longitudinal memory, constraint modeling from historical data, labor-budget forecasting, and skill scarcity detection all require a body of operational data that a new node simply does not have. They are not deferrable because they are difficult — they are deferrable because they are premature.

Optimization and advanced analytics. Lifecycle modeling, feasibility simulation, optimization engines, and automated quality benchmarking represent the system operating at high sophistication. They add value at scale and over time. Deferring them until the foundational systems are stable and the data pipelines are proven is straightforward risk management.

What is proposed as *not* deferrable: any foundational principle. Non-transferability of credits. Ecological flagging as a prerequisite for production. Append-only ledgers in every system. The requirement for certified designs before COS production. The closure of the feedback loop from FRS to CDS. The separation of authority between systems. The argument is that these must be present in every node from the first day of operation — that they are not features but architecture. That argument itself is subject to challenge and ratification by the community.

The deferral list is a starting point for deliberation, not a final determination. If the community's ratification process concludes that a module proposed for deferral is in fact foundational — or that something proposed as foundational can be safely simplified further — the ratified position supersedes what is written here. Those revisions belong in the decision record.

---

4. Critical Data Structures

4.1 Why Data Architecture Comes First

In most software projects, data architecture is treated as an implementation detail — something that emerges from the code as it is written and gets formalized later, if at all. In Integral, this approach would be catastrophic.

The five systems of Integral are not independent applications that happen to share a network. They are deeply coupled components of a single architecture, each consuming data produced by others, each depending on the integrity and shape of what it receives. If COS produces labor event records in a format that ITC cannot reliably consume, contribution accounting breaks. If ITC's credit records do not carry the fields that FRS needs to assess equity patterns, the feedback loop is blind to fairness drift. If OAD's certified designs do not carry the ecological fields that COS needs to attach to materials records, the ecological accounting chain is broken before production begins.

These are not bugs that can be fixed later with a patch. They are structural incompatibilities that require schema migrations, data backfilling, and potentially the invalidation of historical records — exactly the kind of disruption that destroys institutional memory and erodes contributor trust in a young project.

The solution is to define the critical data structures before significant code is written. Not every field needs to be implemented immediately. But every field that the full system will eventually require needs to exist in the schema from day one — even if it is null, even if it is a placeholder, even if the module that populates it has not yet been built.

This is the distinction between what Section 3 describes and what this section describes. Section 3 defines what gets **implemented** in the minimal build — which modules are active, which are deferred, and why. This section defines what gets **defined** in the minimal build — the complete data schemas that all five systems will eventually require, present in full from day one with deferred fields held as typed, nullable placeholders. A module can be deferred. Its data structure cannot be.

One class of fields deserves particular attention: **federation fields**. The system is designed to eventually operate across multiple independent nodes. Every participant ID should be structured as `node-id:participant-id`, not just `participant-id`. Every decision record should include the originating node. Every FRS signal should include geographic scope. Every identifier should be globally unique, not just locally unique. Building these fields in now — even though they will mostly be null in the early build — is the difference between a federation migration that takes days and one that takes months and corrupts historical records.

This section defines the core data contracts between systems. It is written to be readable by non-technical contributors — understanding what data exists and why matters for everyone, not just developers. The process by which this schema design work is completed before development begins is described in Section 5.1.

4.2 The Core Data Contracts Between Systems

A data contract is an agreement between two systems about what information will be exchanged, in what form, and with what guarantees. There are twelve primary cross-system data contracts in the minimal architecture.

OAD → COS: The Certified Design Package — The contract by which a certified design in OAD becomes the authoritative source of truth for a COS production plan. COS cannot initiate production without a certified design, and the design package is the complete specification — labor steps, skill requirements, material quantities, ecological coefficients, and lifecycle assumptions — that COS translates into an executable work breakdown structure.

OAD → ITC: The Design Intelligence Signal — The contract by which OAD certified designs become structured inputs to ITC valuation. ITC cannot compute access costs without OAD's labor-step decompositions, skill tier requirements, material intensity data, and ecological coefficients. OAD flags certified designs to ITC as deployable references at the same time it flags them to COS. When a design is updated or recertified, ITC must receive the updated intelligence to keep access valuations grounded in current design reality.

OAD → FRS: The Design Event Signal — The contract by which OAD design events, ecological assessments, and certification changes become structured inputs to FRS monitoring. FRS cannot assess whether production outcomes are consistent with design assumptions without knowing what those assumptions were. This contract also carries certification status changes — when a design is revoked or superseded, FRS must know.

FRS → OAD: The Operational Recalibration — The contract by which FRS returns real-world evidence to OAD when operational monitoring reveals divergence from certified assumptions. This covers ecological coefficient recalibration, lifecycle revision, and certification review triggers. OAD retains full authority over design decisions — FRS provides the evidence, not the verdict.

COS → ITC: The Labor and Materials Record — The contract by which COS production activity becomes the basis for ITC credit calculations. COS records labor hours, skill tiers, material consumption, tool usage, and peer-verification outcomes. These records are the raw inputs from which ITC computes contribution values and access costs. If this contract is unreliable, ITC has no grounding in physical reality.

COS → FRS: The Operational Signal — The contract by which COS production data becomes the input to FRS diagnostic analysis. This includes throughput rates, material consumption, bottlenecks, maintenance cycles, QA outcomes, and any significant deviation from planned production. FRS cannot monitor system health without a continuous, structured view of what COS is actually producing.

ITC → FRS: The Credit and Access Signal — The contract by which ITC credit balances, decay events, access extinguishments, and equity indicators become inputs to FRS monitoring. FRS reads ITC's ledger as one of its primary signal sources for detecting patterns of access strain, contribution imbalance, or emerging inequity across the system.

FRS → CDS: The Sensemaking Artifact — The contract by which FRS diagnostic findings, scenario models, and recommendations become inputs to CDS governance deliberation. This is the contract that closes the primary operational feedback loop — it is how the system's observable reality reaches the governance process that can act on it.

CDS → FRS: The Governance Signal — The contract by which CDS decision activity — dispatch packets issued, policies updated, governance issues opened and closed — becomes structured input to FRS monitoring. This enables FRS to assess whether implemented decisions are having their intended effects, closing the governance evaluation loop as distinct from the operational feedback loop. Without this contract, FRS would know what the system produces but not whether governance is responding effectively to what FRS reports.

CDS → OAD: The Design Mandate — The contract by which CDS governance decisions become actionable instructions for OAD. This includes authorized design projects, design standards and constraints, certification policy changes, and ecological thresholds that OAD must apply. CDS does not design — it authorizes and constrains. OAD executes within those bounds.

CDS → COS: The Production Mandate — The contract by which CDS governance decisions become actionable instructions for COS. This includes authorized projects, workflow parameters, cooperative formation directives, and any operational constraints that CDS has determined through governance. COS self-organizes within these bounds — the mandate defines the envelope, not the method.

CDS → ITC: The Policy Signal — The contract by which CDS governance decisions become the normative rules that ITC operates within. This includes labor weighting parameters, decay rules, fairness thresholds, access policy modifications, and anti-coercion boundaries. ITC does not set its own policy — CDS does. This contract is how democratic governance translates into the specific parameters that govern contribution recognition and access allocation across the entire node.

4.3 Fields That Must Exist from Day One (Even If Empty)

The following data structures define the minimum schema for each critical object. Fields marked `[DEFERRED]` must exist in the schema but may be null until the module that populates them is built.

A note on relationship to the white paper: the white paper's formal specifications define these objects at a higher level of granularity than is practical for Phase 2 implementation. Where the white paper separates concerns across multiple objects — for example, separating `DesignVersion`, `MaterialProfile`, `EcoAssessment`, `LaborProfile`, and `CertificationRecord` into distinct types — this document consolidates them into single records appropriate for the minimal architecture. These are deliberate Phase 2 simplifications, not errors. The schema field names, enumerations, and type choices below are aligned with the white paper's formal specifications wherever they exist.

The Certified Design Package *(owned by OAD)*

```
CertifiedDesign {
    design_id           // unique identifier, generated at first submission
    version             // semantic version string, e.g. "1.0.0"
    label               // short version label, e.g. "v0.3-bamboo-frame"
    title               // human-readable name
    description         // plain-language description
    authored_by         // list of node-id:participant-id references
    created_at          // timestamp of first submission
    certified_at        // timestamp of certification decision
    certified_by        // List[str] — participant IDs of all panel members who certified
                        // this design. Must be a list, not a single reference, to preserve
                        // the accountable audit trail of the full certification panel.
                        // Matches white paper CertificationRecord schema: certified_by: List[str]
    cds_mandate_ref     // reference to the CDS decision that authorized design work
                        // [devguide extension — not in white paper schema, added for traceability]
    status              // draft | under_review | optimized | ready_for_certification |
                        //   certified | deprecated
                        // [white paper includes optimized and ready_for_certification
                        //  as meaningful intermediate states — do not omit]

    bill_of_materials   // list of MaterialRequirement objects
    production_steps    // ordered list of LaborStep objects
                        // each step carries: name, estimated_hours, skill_tier,
                        //   tools_required, sequence_index, safety_notes
    skill_requirements  // hours_by_skill_tier — breakdown of total estimated hours
                        //   per skill tier across all production steps

```

```
    skill_tier          // low | medium | high | expert
                        // [canonical enum — use consistently across all schemas]

    ecological_flag     // low | medium | high — set during OAD Module 3 assessment
    ecological_notes    // human-authored narrative from ecological checklist
    ecological_detail   // [DEFERRED] structured EcoAssessment data:
                        //   embodied_energy, carbon_intensity, toxicity,
                        //   recyclability, water_use, land_use, repairability

    lifecycle_detail    // [DEFERRED] LifecycleModel data:
                        //   expected_lifetime_years, maintenance_interval_days,
                        //   maintenance_labor_hours_per_interval, disassembly_hours,
                        //   dominant_failure_modes, lifecycle_burden_index

    itc_access_cost     // computed access cost in time credits — set at certification
    itc_access_cost_method  // simple_embodied | [future methods]

    superseded_by       // reference to superseding design_id, null if current
    design_lineage      // reference to prior version design_id, null if first version
    node_id             // originating node — federation-aware from day one
    federation_hash     // [DEFERRED] cryptographic hash for cross-node verification
    commons_visible     // boolean — whether published to the commons
}
```

The Labor Event Record *(owned by COS, consumed by ITC and FRS)*

The white paper separates raw labor capture (`LaborEvent`) from weighted contribution recognition (`WeightedLaborRecord`). This is architecturally significant: the labor event is value-neutral until weighting is applied. The devguide consolidates these into a single record for Phase 2 simplicity, but the boundary between raw capture and weighted output must be preserved in the field structure. Note that the white paper uses a single `hours` field on `LaborEvent` — the `hours_verified` field below is a devguide extension for Phase 2 convenience. The `WeightedLaborRecord` fields are consolidated here but belong conceptually to ITC Module 2.

```
LaborEvent {
    event_id            // unique identifier
    timestamp           // when the labor was completed
    participant_id      // node-id:participant-id — globally unique
    coop_id             // which cooperative this labor occurred within
    task_ref            // reference to the COS Task object
    production_plan_ref // reference to the COS ProductionPlan
    design_ref          // reference to the OAD CertifiedDesign and version

    hours               // actual hours worked, as reported
    hours_verified      // hours confirmed by peer verification
                        // [devguide extension for Phase 2 convenience]
    skill_tier          // low | medium | high | expert
                        // [canonical enum]

    context             // structured weighting context signals:
                        //   urgency_score, eco_sensitive, scarcity_factor, etc.
                        // [white paper: context dict is primary input to Module 2 —
                        //  must not be omitted]

```

```
    verification_status // pending | verified | disputed
    verified_by         // list of node-id:participant-id of verifiers
    verified_at         // timestamp of verification

    // Weighting outputs — populated by ITC Module 2
    // [white paper: these belong to WeightedLaborRecord, consolidated here for Phase 2]
    itc_weight_multiplier   // final bounded weight multiplier applied
    itc_weight_breakdown    // dict: {"skill_factor": 1.3, "scarcity_factor": 1.1, ...}
    itc_weighted_hours      // hours * weight_multiplier
    itc_credits_issued      // credits generated — computed by ITC

    ecological_flag     // inherited from the design's production step
    ecological_detail   // [DEFERRED] detailed ecological attribution

    node_id             // which node this event occurred at
    federation_hash     // [DEFERRED] cryptographic hash for federation integrity
}
```

The Materials Consumption Record *(owned by COS, consumed by ITC and FRS)*

This schema is a devguide construction derived from white paper requirements for COS production data. No direct equivalent formal schema exists in the white paper, but the fields are consistent with what the white paper specifies COS must generate.

```
MaterialConsumptionEvent {
    event_id            // unique identifier
    timestamp           // when consumption occurred
    task_ref            // reference to the COS Task
    production_plan_ref // reference to the COS ProductionPlan
    design_ref          // reference to the OAD CertifiedDesign and version

    material_id         // what was consumed
    material_name       // human-readable name
    quantity_consumed   // numeric quantity
    unit                // unit of measurement
    quantity_remaining  // inventory level after consumption

    ecological_flag     // low | medium | high — from OAD design
    ecological_impact_index  // [DEFERRED] numeric EII from coefficient engine

    source              // internal | external_dependency
    external_source_ref // [DEFERRED] if external, reference to supplier/node

    node_id             // which node this occurred at
    federation_hash     // [DEFERRED] cryptographic hash
}
```

The ITC Account Record *(owned by ITC, consumed by FRS)*

The white paper's ITC formal specification uses `ITCAccount` as the standing balance record. This schema aligns with that object. The white paper does not define a separate `CreditBalance` object — the account record carries both the current balance and the active decay configuration.

```
ITCAccount {
    account_id          // unique identifier
    participant_id      // node-id:participant-id — globally unique
    node_id             // which node issued these credits

    balance             // current credit balance, numeric
    total_earned        // total credits ever issued
    total_redeemed      // total credits extinguished through access
    total_decayed       // total credits lost to decay

    active_decay_rule_id    // reference to the CDS-ratified DecayRule in effect
    last_decay_applied_at   // timestamp of most recent decay calculation
    last_contribution_at    // timestamp of most recent verified labor event
                            // [required for decay grace window logic — avoids full
                            //  ledger scan to determine whether grace period applies.
                            //  updated by ITC Module 1 on each verified LaborEvent]
    last_access_at          // timestamp of most recent ITC redemption or access event
                            // [required for FRS access pattern monitoring.
                            //  updated by ITC Module 5 on each redemption]

    // DecayRule (inline summary — full record lives in CDS policy store)
    decay_half_life_days        // exponential half-life beyond grace window
    decay_inactivity_grace_days // no decay within this window
    decay_min_balance_protected // small protected floor, if any
    decay_max_annual_fraction   // CDS-set safety bound on annual decay

    need_adjustment_active  // boolean — whether a need-based override is active
    need_adjustment_ref     // reference to CDS policy record authorizing adjustment

    transferable        // always false — enforced architecturally, never policy
                        // [this field exists as a named constraint, not a toggle.
                        //  any attempt to implement transfer requires intentionally
                        //  changing a named architectural boundary]
    federation_scope    // local | [DEFERRED: cross-node recognition scope]

    last_updated_at     // timestamp of most recent change to any field
}
```

The ITC Ledger Entry *(owned by ITC, consumed by FRS)*

The white paper separates `LaborEvent` (raw hours) from `WeightedLaborRecord` (weighted credits) as distinct objects. In the ledger, both values must be preserved for labor-related entries so that FRS can audit weighting patterns over time without reconstructing them from two separate records. The `amount` field carries the net credit change; `raw_hours` and `weighted_credits` provide the labor-specific detail for entries where that distinction is meaningful.

```
ITCLedgerEntry {
```

```
    entry_id            // unique identifier
    timestamp           // when this entry was created
    participant_id      // node-id:participant-id — globally unique
    node_id             // which node

    entry_type          // labor_event_recorded | labor_weight_applied |
                        // itc_credited | itc_decayed |
                        // access_value_quoted | access_redeemed |
                        // equivalence_band_applied |
                        // ethics_flag_created | ethics_flag_resolved |
                        // policy_updated
                        // [white paper canonical enum — use exactly]

    amount              // numeric — net credit change for this entry (positive for
                        // credits issued or restored, negative for deductions);
                        // for labor entries this equals weighted_credits
    raw_hours           // nominal hours worked, unweighted — populated for labor
                        // entry types only, null for decay/redemption/policy entries
                        // [white paper separates LaborEvent from WeightedLaborRecord;
                        //  storing both here enables FRS audit of weighting inflation
                        //  over time without reconstructing from two separate records]
    weighted_credits    // credits after skill/context weighting — equals amount for
                        // labor entries; null for non-labor entry types
                        // [FRS uses the ratio of weighted_credits to raw_hours
                        //  across the ledger to detect systematic weighting drift]
    balance_after       // balance snapshot after this entry

    related_ids         // dict of references to related objects:
                        //   e.g. {"event_id": "...", "item_id": "...", "coop_id": "..."}
    details             // JSON-compatible payload with entry-specific data

    authorized_by       // participant or system that authorized this entry
    notes               // optional human-authored explanation
}
```

The FRS Signal Packet *(owned by FRS, consumed by CDS)*

The white paper's `SignalPacket` is an abstract normalized bundle of `SignalEnvelope` objects. The structure below is a Phase 2 implementation that adds human-readable summary sections on top of that abstract model. The summary fields make the packet directly usable for governance deliberation without requiring CDS to query raw envelopes. This is a devguide extension, not a white paper schema.

```
FRSSignalPacket {
    packet_id           // unique identifier
    node_id             // which node this covers
    period_start        // start of the period covered
    period_end          // end of the period covered
    generated_at        // when this packet was produced
    generated_by        // participant or system that produced it
    prev_packet_ref     // reference to prior period packet for comparison
    packet_hash         // [DEFERRED] cryptographic chaining hash

```

```
    labor_summary {
        total_events                // count of labor events in period
        total_hours                 // sum of raw hours
        total_weighted_credits      // sum of weighted_credits across labor entries
                                    // [enables period-level weighting ratio analysis]
        total_hours_verified        // sum of verified hours
        verification_rate           // ratio of verified to total
        events_by_skill_tier        // breakdown by skill tier
                                    // [uses low | medium | high | expert]
        overrun_tasks               // tasks where actual significantly exceeded estimate
    }

    materials_summary {
        total_consumption_events
        consumption_by_material     // breakdown by material_id
        ecological_flag_counts      // count of low | medium | high events
        external_dependency_count
        low_inventory_flags
    }

    itc_summary {
        total_credits_issued
        total_decay_applied
        total_access_extinguished
        active_participant_count
        balance_distribution        // [DEFERRED] statistical distribution
        need_adjustments_active
    }

    qa_summary {
        total_qa_events
        failures
        failure_rate
        failures_by_design          // which OAD design_id/version is implicated
    }

    oad_summary {
        new_certifications
        design_version_updates
        ecological_assessment_changes
        certification_revocations
        coefficient_recalibrations_pending
    }

    cds_summary {
        decisions_dispatched
        policies_updated
        open_governance_issues
        decision_records_merged
    }
    // cds_summary enables FRS to assess whether implemented decisions
    // are producing their intended effects — closing the governance
    // evaluation loop as distinct from the operational feedback loop

    findings            // list of DiagnosticFinding objects
```

```
    recommendations     // list of Recommendation objects
    cds_submission_ref  // reference to CDS issue created from this packet
}
```

The Diagnostic Finding *(owned by FRS)*

```
DiagnosticFinding {
    finding_id          // unique identifier
    packet_ref          // reference to the FRSSignalPacket
    node_id             // which node
    detected_at         // timestamp

    finding_type        // ecological_overshoot_risk |
                        // material_scarcity_trend |
                        // labor_imbalance_or_burnout_risk |
                        // throughput_bottleneck_persistent |
                        // quality_reliability_drift |
                        // dependency_fragility_increase |
                        // access_inequity_detected |
                        // proto_market_or_coercion_risk |
                        // governance_overload_or_capture_risk |
                        // data_integrity_anomaly |
                        // other
                        // [white paper canonical enum]

    severity            // info | low | moderate | high | critical
    scope               // local | node | regional | federation
                        // [white paper field — scope determines routing and
                        //  escalation path, must not be omitted]
    persistence         // transient | emerging | persistent | structural
                        // [white paper includes structural as a fourth state]
    confidence          // low | medium | high

    summary             // one-sentence plain-language description
    rationale           // explanation of why this is a finding
    evidence_refs       // list of references to supporting records
    indicators          // dict of named metric values that triggered this finding

    target_system       // CDS | OAD | COS | ITC | FED
                        // [FED is used for federation-level findings]
    requires_cds        // boolean — whether CDS deliberation is required
}
```

The Recommendation *(owned by FRS)*

```
Recommendation {
    recommendation_id   // unique identifier
    finding_ref         // reference to the DiagnosticFinding
    constraint_model_ref // reference to the ConstraintModel that informed this
                        // [links recommendation to the scenario modeling
```

```
                        //  that generated it]
    node_id             // which node
    created_at          // timestamp

    target_system       // CDS | OAD | COS | ITC | FED
    recommendation_type // design_review_request |
                        // workflow_stress_alert |
                        // valuation_drift_flag |
                        // training_priority_signal |
                        // material_substitution_prompt |
                        // dependency_risk_alert |
                        // policy_review_prompt |
                        // monitoring_directive |
                        // federated_learning_share |
                        // other
                        // [white paper canonical enum]

    severity            // inherited from finding
    scope               // inherited from finding
    confidence          // low | medium | high

    summary             // plain-language description of suggested action
    rationale           // why this action is suggested
    payload             // structured details relevant to the target system

    status              // pending | acknowledged | accepted | rejected | superseded
    response_ref        // reference to the action taken in response, if any
}
```

4.4 API Boundaries and Inter-System Interfaces

In the minimal build, the five systems may share a single codebase and a single database, with system boundaries enforced by module separation and access control rather than physical service boundaries. This is acceptable for a first node. What is not acceptable is designing the code as if the boundaries do not exist — writing COS functions that directly manipulate ITC records, or FRS logic that writes to CDS tables.

Even in a monolithic initial implementation, every cross-system data access must go through a defined interface function that could later become an actual API endpoint without requiring any change to the calling code. This discipline has a specific name: **designing for seam points**. Seams are where the system will eventually be divided into independent services as it scales. Building them cleanly from the beginning means the transition from a single-node monolith to a federated multi-service architecture is a deployment change, not a rewrite.

The interface function signatures defined below are devguide architectural decisions, not transcriptions of the white paper. The white paper specifies data structures and module behaviors but does not define the inter-system API layer. These signatures represent the minimum interface surface needed to implement the data contracts in Section 4.2. They are starting points for community ratification — working groups should treat them as proposals subject to revision through the standard governance process before implementation begins.

OAD → COS Interface

```
get_certified_design(design_id, version) → CertifiedDesign | NotFound
list_certified_designs(filters) → List[CertifiedDesign]
verify_design_status(design_id, version) → { certified: bool, status: string }
```

OAD → FRS Interface

```
get_design_events(node_id, period_start, period_end) → List[DesignEvent]
get_ecological_assessment(design_id, version) → EcologicalAssessment
get_certification_changes(node_id, period_start, period_end) → List[CertificationChangeEvent]
```

DesignEvent covers new submissions, version updates, and certification status changes. *EcologicalAssessment* carries material profiles, ecological flags, lifecycle assumptions, and the coefficients FRS may need to recalibrate. *CertificationChangeEvent* records every certification, supersession, or revocation with the reason and the FRS finding that triggered it.

FRS → OAD Interface

```
submit_coefficient_recalibration(design_id, version, delta: CoefficientDelta, evidence_ref) → AcknowledgementRecord
submit_lifecycle_revision(design_id, version, revised_assumptions: LifecycleAssumptions, evidence_ref) → AcknowledgementRecord
submit_certification_review_request(design_id, version, finding_ref, severity) → ReviewRecord
```

These functions represent FRS's authority to push corrective signals back into OAD based on operational reality. `submit_coefficient_recalibration` adjusts ecological coefficients when real-world degradation, scarcity, or ecosystem strain diverges from modeled values. `submit_lifecycle_revision` updates durability and maintenance assumptions when deployed goods perform differently than certified. `submit_certification_review_request` initiates a formal review request for a design's certification status — OAD receives the request and retains full authority over whether and how to act on it. FRS provides the evidence and initiates the review; it does not direct the outcome.

Critical boundary: `submit_certification_review_request` does not halt production. It submits a non-executive recommendation that routes to OAD as a flagged concern. Any decision to suspend, revoke, or supersede a certification must pass through OAD review and ultimately through CDS deliberation before it has any operational effect. FRS flags the divergence. It does not pull the trigger. Production continues unless and until CDS ratifies a decision to the contrary through the standard governance process. This boundary is what prevents FRS diagnostic authority from becoming a form of algorithmic shadow governance.

All three functions require an evidence reference — a finding from FRS Module 2 — and are recorded permanently in both the FRS and OAD ledgers.

COS → ITC Interface

```
get_labor_events(node_id, period_start, period_end) → List[LaborEvent]
get_material_events(node_id, period_start, period_end) → List[MaterialConsumptionEvent]
get_production_summary(plan_id) → ProductionSummary
```

COS → FRS Interface

```
get_operational_signal(node_id, period_start, period_end) → OperationalSignalData
get_qa_events(node_id, period_start, period_end) → List[QAEvent]
```

ITC → FRS Interface

```
get_credit_summary(node_id, period_start, period_end) → ITCPeriodSummary
    // ITCPeriodSummary includes: total credits issued, decay applied,
    // access extinguishments, active participant count, and balance distribution.
    // Also returns per-account last_contribution_at and last_access_at timestamps
    // [these fields are read directly from ITCAccount — FRS must never
    //  reconstruct them by scanning the full ledger. They are the authoritative
    //  source for decay grace window state and access pattern monitoring]

get_ledger_entries(node_id, period_start, period_end) → List[ITCLedgerEntry]
    // Returns append-only ledger entries for the period, including
    // raw_hours and weighted_credits fields on labor entry types

get_balance_distribution(node_id) → BalanceDistribution
```

FRS → CDS Interface

```
submit_signal_packet(packet: FRSSignalPacket) → IssueReference
get_recommendation_status(recommendation_id) → RecommendationStatus
```

CDS → FRS Interface

```
get_governance_summary(node_id, period_start, period_end) → GovernanceSummary
  // Returns dispatch count, policy changes, open issues, and DRs merged
  // Enables FRS to monitor whether governance decisions are producing effects
```

CDS → All Systems: Policy and Mandate Interface

```
get_active_policies(system, node_id) → List[PolicyRecord]
get_itc_policy_snapshot(node_id) → ITCPolicySnapshot
  // Returns the current CDS-ratified parameters that bound ITC behavior:
  // weighting limits, decay rule in effect, fairness thresholds,
  // access policy bounds, and anti-coercion rules.
  // ITC Module 9 uses this snapshot to synchronize with CDS policy
  // without ITC becoming a policy authority itself.
get_production_mandate(mandate_id) → ProductionMandate
notify_dispatch(dispatch_packet: DispatchPacket) → AcknowledgementRecord
  // DispatchPacket carries system-specific payloads:
  //   oad_flags    — design updates and standards changes for OAD
  //   tasks        — workflow and cooperative formation directives for COS
  //   itc_adjustments — weighting and access rule changes for ITC
  //   frs_monitors — monitoring parameters and success metrics for FRS
```

Every interface function carries a version identifier. When an interface changes in a way that is not backward compatible, the version number increments and the prior version remains available until all consumers have migrated. This rule applies from the first interface ever written. In a federated architecture where different nodes may be running different software versions, interface backward compatibility is not a convenience — it is what makes federation possible at all.

---

5. The Phased Expansion Path

Integral is developed through four broad phases. These phases are not rigid technical gates or fixed schedules. They are practical stages of maturation, each building on the prior one. The completion of each phase is determined by the relevant working groups based on what has been learned, what has been built, and whether the project is ready to move forward responsibly.

The purpose of this section is not to specify every procedural detail in advance. Those details belong in the governance and development coordination sections that follow. What matters here is the overall developmental logic: first the project clarifies how it will govern itself and what it is building, then it builds the minimum viable system, then it tests that system under simulated and virtual conditions, and finally it brings that system into real-world node operation.

```
The Phased Expansion Path

├── Phase 1: Governance and System Definition
│   ├── Review white paper and architecture
│   ├── Define decision-making structure
│   ├── Establish CDS as initial priority
│   └── Define Minimum Viable System scope
│
├── Phase 2: Minimum Viable System Development
│   ├── Build CDS for real project decisions
│   ├── Develop OAD, ITC, COS and FRS to minimum spec
│   ├── Establish working loop across all 5 systems
│   └── Defer advanced features unless necessary
│
├── Phase 3: Simulation and Virtual Node Testing
│   ├── Run simulation environments
│   ├── Virtual node testing with real participants
│   ├── Model Interface Cooperative structures
│   └── Identify failures and assess deployment readiness
│
└── Phase 4: Real-World Deployment
    ├── Pilot node operation under live conditions
    ├── Govern real activity via MVS
    ├── Activate Interface Cooperatives
    ├── Begin inter-node development
    └── Iterate and grow the network
```

5.1 Phase 1 — Governance and System Definition

Phase 1 establishes the project's conceptual and procedural foundation. At this stage, Integral is not yet a working software system. The task is to create the conditions under which one can be built coherently and legitimately. The development community clarifies how it will make decisions, how contributors will coordinate, and what the first real build is actually intended to include.

This phase also focuses on reviewing the integrity of the white paper and the proposed architecture as a whole. The goal is not to treat the existing theory as settled, but to identify ambiguities, tensions, and open design questions before implementation begins. The proposed minimum viable system is defined during this period: the smallest version of Integral that can still demonstrate the core architectural principles honestly.

Within that definition, the CDS is established as the initial development priority. This is not because governance is more important than the other systems in the abstract, but because the project requires a legitimate decision-making structure before more complex architectural and implementation choices can be made with confidence. Phase 1 therefore ends with a clearer shared understanding of the architecture, the development path, and the minimum viable system the project intends to build first.

A Phase 1 deliverable: the schema design exercise. Before Phase 2 development begins, a deliberate schema design exercise must be completed. This is not glamorous work and it does not produce running software — but it is arguably the highest-leverage technical work in the entire project. Working group by working group, contributors sit with the white paper's full module specifications and the data contracts in Section 4, and ask of every object and every field: will the full system eventually need this, and if so, is it represented in the schema? The output of this exercise is a ratified, complete schema for each system — with all deferred fields present as typed, nullable placeholders — before the first line of production code is written. Teams that skip this step and build quickly against an incomplete schema will produce a local success and a future migration problem. The append-only ledgers that form the system's institutional memory will contain records in an incompatible format. That is not a bug that gets fixed — it is a constraint that gets carried indefinitely or resolved at very high cost. The schema design exercise is the insurance against that outcome. Its completion is a condition for Phase 2 to begin.

5.2 Phase 2 — Minimum Viable System Development

Phase 2 is the construction phase for the minimum viable system itself. The project moves from conceptual preparation into implementation, building the first operational form of all five systems in a deliberately minimal but architecturally faithful way. The emphasis is not on full sophistication, but on establishing a real, working loop across CDS, OAD, ITC, COS, and FRS.

Within this build, CDS receives special priority because it must begin guiding the development process as the contributor community and architectural workload become more complex. The goal is not necessarily a fully mature governance engine at this stage, but a stable and transparent CDS capable of handling real project decisions in a structured and accountable way. In parallel, the other four systems are developed to the point where they can meet the minimum functional requirements of the first complete system loop.

This phase should be understood as the creation of a coherent first architecture, not the completion of the system in its final form. Sophisticated analytics, federation-dependent capabilities, and advanced optimization remain deferred unless they are shown to be necessary. The purpose here is to build a functioning minimum viable system that preserves the project's foundational principles and is ready to be tested as a whole.

What Constitutes Completion of the Minimal Viable System

Completion of the Minimal Viable System (MVS) does not refer to scale, stability, or production readiness. The goal of Phase 2 is architectural validation.

The MVS is considered complete when all five systems operate together in a closed loop and the following conditions are demonstrated:

1. A contributor can introduce an issue or proposal into CDS.
2. CDS can produce a ratified decision and dispatch a corresponding decision packet affecting another system.
3. OAD designs can be referenced in COS production workflows.

4. COS production events generate verifiable contribution records.
5. ITC credits are issued for verified contribution and later extinguished through access events.
6. FRS reads system activity and returns feedback signals to CDS for governance awareness.

The number of participants required to demonstrate this loop is not fixed; a small testing community is sufficient. The objective is not scale but verification that the architecture functions coherently when its systems interact.

Integration Strategy for the Minimum Viable System

Although the five Integral systems have strong dependencies, they are not developed in a fixed sequential order. Development instead follows a **vertical slicing strategy**.

The goal is to create a thin but complete operational path through all five systems as early as possible. Each system begins with the smallest possible set of functions required to participate in the core loop. This allows integration to occur early and repeatedly, preventing the architectural drift that occurs when subsystems are developed in isolation.

Development therefore proceeds by expanding capability across all systems together rather than completing any one system independently. No subsystem should reach deep functional complexity while other systems remain unimplemented or disconnected from the loop.

The first successful slice should allow the following minimal path:

```
proposal introduced in CDS
→ certified design referenced in OAD
→ production workflow executed in COS
→ contribution recorded in ITC
→ operational signals analyzed in FRS
→ feedback returned to CDS
```

Once this loop operates successfully, subsequent development expands module depth and capability across the systems together while preserving continuous integration across the full architecture.

5.3 Phase 3 — Simulation and Virtual Node Testing

Once the minimum viable system has been built, Phase 3 focuses on testing, validation, and learning. The architecture is run through simulation environments in order to observe how the five-system loop behaves under different conditions, loads, and constraints. This includes not only technical simulation, but broader systems testing intended to surface failure modes, ambiguities, and structural weaknesses before real-world deployment.

Phase 3 also includes virtual node operation involving real participants using real governance, contribution, and coordination criteria, but without relying on physical production infrastructure. This allows the project to test the social and procedural realities of the architecture alongside the technical ones. The point is not merely to prove that the software runs, but to determine whether the system behaves intelligibly and coherently when actual people use it in something approximating real operation.

This is also the stage in which transitional real-world bridge structures, including the logic of Interface Cooperatives, can be modeled and simulated in a controlled way. Since these structures become relevant when Integral interacts with supply chains, legal entities, and the remaining market environment, Phase 3 is the right place to test their role conceptually and operationally before they are instituted in practice. By the end of this phase, the project should have a much clearer understanding of what parts of the architecture hold, what requires revision, and what is needed for real deployment.

5.4 Phase 4 — Real-World Deployment

Phase 4 introduces the minimum viable system into the physical world through pilot node operation. At this point, the project moves beyond simulation and virtual governance into actual cooperative activity under real constraints. The purpose is to determine whether the architecture can function as an operational socio-technical system rather than only as a modeled one.

In this phase, the minimum viable system is used to govern real activity, organize production, record contribution, and generate feedback under live conditions. Interface Cooperatives become relevant here as transitional structures for handling the system's interaction with external supply chains, legal requirements, and other real-world dependencies that cannot yet be fully internalized by the node. Their role should remain complementary and transitional, supporting the node's operation without displacing the internal cybernetic logic of Integral itself.

Phase 4 also includes the beginning of inter-node development as real communities form, learn, and begin coordinating beyond a single node. This does not require a fully mature federated architecture at the outset. It means that node-to-node cooperation, shared problem solving, and network functionality begin to emerge in practical form as real-world experience accumulates. From this point forward, development proceeds iteratively: strengthening the functionality of individual nodes, improving the architecture where reality exposes its limits, and expanding the network carefully as the project earns its next layer of complexity.

NOTE: Although the phases are presented sequentially, development is expected to move iteratively between them. Insights from simulation or early node operation may reveal architectural weaknesses, governance problems, or design assumptions that require revision. When this occurs, the project may return to earlier phases in order to refine the architecture before proceeding. The phased structure therefore describes a developmental path rather than a one-directional pipeline.

---

6. Contributing to Integral

Integral is not a traditional software project. It is an attempt to design and implement an integrated socio-technical system — one that requires theory, software, governance practice, ecological insight, and critical scrutiny simultaneously. The contributor community therefore needs to be as integrated as the system itself.

This section explains how people enter the project, where work happens, and how contributions move through the development process.

6.1 Types of Contributors

Integral welcomes many kinds of contributors. Not everyone writes code, and the project cannot succeed if only software developers participate. The following five types describe common ways people engage with the project. These are not roles or ranks — they are starting points. Many participants move between them over time.

Builders — Developers, engineers, and systems architects who translate specifications into working software. Builders typically contribute within one of the five system working groups, implementing modules, interfaces, and infrastructure. Read the development guide and relevant white paper sections before beginning implementation. The white paper's pseudocode and system descriptions are conceptual starting points, not finished designs.

Thinkers — Researchers, theorists, economists, governance scholars, and domain experts who engage with the architecture at the conceptual level. Thinkers examine whether the system is logically sound, whether its assumptions hold under scrutiny, and whether it addresses the problems it claims to solve. Well-argued critique is among the most valuable contributions the project receives — identifying a flaw in a mechanism early is often more valuable than implementing a new feature.

Practitioners — People with real experience running cooperative organizations, mutual aid networks, commons governance systems, or community-based initiatives. Practitioners evaluate whether the system would function under real social conditions. Their insights into failure modes, coordination friction, and governance dynamics are essential. Integral must work not only in theory but in lived cooperative practice. Practitioners with structured facilitation experience are particularly needed for the CDS Module 9 escalation path, where irreducible value conflicts require high-bandwidth human deliberation — and eventually,

Syntegrity sessions — that no algorithm can substitute for.

Makers — Writers, designers, communicators, and documentarians who make the project understandable and accessible. Makers create documentation, onboarding materials, visual explanations, and public communication that allow new contributors and future node participants to understand how the system works. Clear communication is not peripheral to a project of this complexity — it is load-bearing.

Explorers — People interested in the project who are unsure where they fit yet. Explorers are not a lesser category — they are often future Builders, Thinkers, Practitioners, or Makers who need context and a path to begin. Start with the white paper introduction and the development guide, then join project discussions when ready.

6.2 Where Work Happens

Integral development takes place across two platforms: **GitHub** and **Discord**. Each serves a distinct and non-interchangeable purpose.

GitHub is the durable record of the project. All important work eventually lands there — architecture discussions, technical issues, implementation work, governance decisions, decision records, and research contributions. GitHub is where the project's institutional memory lives.

Discord is the real-time collaboration space. It supports conversation, coordination, early exploration of ideas, working group discussion, and community interaction. Discord is where contributors find each other and develop ideas before those ideas are formally recorded on GitHub.

The relationship between the two platforms follows a single governing rule: **Discord begins conversations. GitHub preserves decisions.** Nothing important should exist only in Discord. When a discussion produces a conclusion, a proposal, or a resolution, it must be captured in GitHub.

GitHub Repositories

Category	Repository	Purpose
Documents	`integral/whitepaper`	Canonical architecture, critique, and revisions
	`integral/devguide`	This document and its living updates
	`integral/website`	Site content and design
Systems	`integral/cds`	Collaborative Decision System *(priority)*
	`integral/oad`	Open Access Design
	`integral/cos`	Cooperative Organization System
	`integral/itc`	Integral Time Credits
	`integral/frs`	Feedback & Review System
Infrastructure	`integral/specifications`	Interface contracts and data schemas
	`integral/decisions`	Append-only governance record (DR-001+)

Discord Channels

Category	Channel	Purpose
Welcome	`#start-here`	Orientation and pinned links — read-only
	`#announcements`	Write-restricted, project-wide only — read-only
	`#introductions`	New contributor introductions — forum
System Development	`#integral-cds`	CDS module discussion — forum, priority working group
	`#integral-oad`	OAD module discussion — forum
	`#integral-cos`	COS module discussion — forum
	`#integral-itc`	ITC module discussion — forum
	`#integral-frs`	FRS module discussion — forum
	`#specifications`	Interface contracts and schema architecture — forum
Theory & Docs	`#white-paper`	White paper discussion and critique — forum
	`#dev-guide`	Development guide discussion — forum
	`#research`	Systems theory, cybernetics, cooperative economics — forum
Node Community Focus	`#applied-technology`	Node-relevant efficiency technologies — forum
	`#foundational-projects`	Foundational node project ideation — forum
	`#proto-node-dev`	Proto-node development and legal frameworks — forum
Project Management	`#governance-proposals`	Development workflow and governance methodology — forum
	`#critical-issues`	Immediate operational problems — text
Move to GitHub	`#ready-to-propose`	Structured proposal staging ground — forum
Platforms	`#website`	Website and future node online infrastructure — forum
	`#mobile-app`	Mobile application ideation and planning — forum

6.3 The Development and Design Processes

Work within Integral operates across two distinct but related processes: a **development process** and a **design process**. Understanding the distinction matters because conflating them produces confusion about what kind of decision is being made and through what channel it should move.

The **development process** concerns the internal governance of the build itself. It addresses questions about how the contributor community organizes and coordinates: how working groups are structured, how repositories are managed, how decisions about workflow and tooling are made, and how the project governs its own operation as it grows. The development process is not fixed — it is itself subject to revision as the project advances and the community's needs change. Because it is essentially a governance question, decisions within the development process should reflect the same consensus principles that Integral's CDS is designed to

embody. In the early phases, this operates as a provisional precursor to the CDS. For this reason, CDS is established as the priority system in development — so that as the contributor community grows in size and complexity, a functioning governance mechanism is already available to guide it.

The **design process** concerns the integrity of Integral's system architecture and its constituent elements. It proceeds logically from analysis and critique of the white paper, through decisions about module construction, interface contracts, data schemas, implementation sequencing, and technology choices. The design process is where the architecture is stress-tested, refined, and progressively specified into working software. It is the process by which the theoretical specification becomes a real system — and where the inevitable gap between the two is honestly confronted and worked through.

Both processes run in parallel throughout the project's life. Neither is subordinate to the other. A decision about how the contributor community makes decisions (development process) is just as consequential as a decision about how the ITC computes a credit value (design process) — and both require the same commitment to structured deliberation, transparent reasoning, and recorded outcomes.

6.4 Onboarding, Access, and Trust

At the outset of this project there is, by necessity, a single administrative authority — the trustee — who has established the project and holds full administrative access to all repositories and platforms. All access expansion proceeds from there, and is earned, not assumed.

Applying to Contribute

Both GitHub and Discord are publicly visible. Anyone can read repositories, browse decision records, and observe working group channels without an account. Participation — posting, opening issues, engaging in governance — requires approval.

Applications are submitted through a short form on the project website: integralcollective.io/application.html. The form takes approximately 15 minutes to complete and asks open-ended questions about what brought the applicant to the project and what they hope to contribute. There are no credential requirements. The filter is seriousness of engagement, not expertise or background. Applications are reviewed by the trustee and, as the project grows, by a small group of established contributors. Applicants will receive a response within one week, max.

Upon approval, applicants receive a Discord role granting posting permissions and a GitHub invitation granting the ability to open issues and comment. This is the baseline access level for all new contributors.

Access Tiers

Access is simple at the start and expands only as trust is demonstrated through consistent, quality engagement. Time alone does not earn expanded permissions — a visible record of serious contribution does.

GitHub operates on two tiers initially:

- *Contributor* — can open issues, comment, participate in discussion across all repositories, and fork any repository to develop work independently. Forking does not require special permissions — it is the standard working method for new Builders. Fork the relevant repository to your own GitHub account, develop your work there, and submit a pull request when it is ready for review. Merge authority to the main branch remains with the trustee.
- *Trusted contributor* — can push directly to feature branches within their working group's repository. Granted individually by the trustee as a track record is established.

Merge authority to the main branch of any repository is held exclusively by the trustee until a core team has been built and trust has been sufficiently demonstrated to delegate it. This is the last permission granted, not an early one.

Discord operates on a single approved-member role at the outset. Distinctions between working group leads and general members can be added when the community is large enough to warrant them. Complexity in access structure is introduced only when it is genuinely needed.

The Governing Principle

Access expands as trust is demonstrated and contracts when that trust is broken or abandoned. The record that establishes trust lives in GitHub — in the quality of issues opened, the rigor of critiques offered, and the consistency of follow-through on commitments made. That record is public, permanent, and append-only, for the same reasons every other ledger in this project is.

Access may be revoked under two circumstances: prolonged inactivity, where a contributor has ceased meaningful engagement without notice for an extended period; or bad faith conduct, where a contributor has acted in deliberate contradiction of the project's principles — through persistent obstruction, misrepresentation, or actions intended to undermine the integrity of the governance process or codebase. Revocation is an administrative decision made by the trustee, or in later stages by the core team, and is recorded.

6.5 Workflow and Consensus

One of the structural advantages of this project is that it is oriented toward a common technical goal rather than toward the resolution of cultural or ideological differences. While the decisions involved are genuinely complex, shared commitment to the end goal produces a natural convergence over time — variety reduces as contributors develop a common understanding of what the system requires, and discussion tends toward technical refinement rather than open-ended debate. This does not eliminate disagreement, but it gives disagreement a productive direction.

Any contributor can initiate a change — to the architecture, to a module specification, to the development process itself. The path from initial concern to ratified change follows a consistent sequence, regardless of where it originates or what it addresses.

The General Workflow

Step	Platform	Action
1. Discussion	Discord	Concern raised in relevant working channel; mature proposals staged in `#ready-to-propose`
2. Issue	GitHub	Filed in relevant repository using appropriate template
3. Deliberation	GitHub issue thread	Assumptions challenged, alternatives considered, blocking objections raised & addressed
4. Proposal	GitHub issue or document	Specific change articulated and opened for final review
5. Consensus — GitHub issue thread	GitHub issue thread	No unresolved blocking objections — ratification noted
6. Pull Request	GitHub	References issue, reviewed for correctness, merged by trustee
7. Decision Record	`integral/decisions`	Append-only, immutable record of ratified change

1. Discussion — Discord All substantive engagement begins in conversation. A concern is raised, an idea is floated, or a problem is identified — typically in the relevant working channel in System Development, Theory & Docs, or Project Management. This stage is informal and exploratory. Its purpose is to establish whether an idea has enough substance and support to warrant formal development. Not every Discord discussion becomes a GitHub issue, and that is by design. Discord is the filter, not the record.

2. Issue — GitHub When a discussion produces something concrete — a specific problem to solve, a proposed change, a question requiring a decision — it is captured as a GitHub issue in the relevant repository. An issue is not a proposal for a change; it is a structured description of a problem or question that may lead to one. It is the point at which an informal concern becomes a formal work item: visible to all contributors, permanently recorded, and open for deliberation.

A well-formed issue clearly states what the concern is, why it matters, which systems or modules it touches, and what is already known about the range of possible responses. It does not need to arrive with a solution — but it does need to arrive with enough clarity that other contributors can engage with it meaningfully. The discipline of writing a clear issue is itself a useful test of whether a concern is ready to move forward.

The repositories include issue templates for the most common issue types. The five code repositories (cds, oad, cos, itc, frs) each carry four templates: architectural concerns, module proposals, interface and schema changes, and documentation requests. The specifications repository carries four templates specific to its function: architectural concerns, interface and schema changes, schema design questions, and documentation requests. The devguide and whitepaper repositories each carry two templates: proposals and corrections. The website repository carries a single site issue template. White paper critique has no template in the code or specifications repositories — all challenges to the white paper's specifications or assumptions belong in `integral/whitepaper`, regardless of which system surfaced the concern, so that architectural critique remains consolidated and visible across the whole contributor community.

Issues are filed in the repository most directly relevant to their subject. An architectural question about the ITC belongs in `integral/itc`. A proposed change to an interface contract belongs in `integral/specifications`. A governance or process question belongs in `integral/decisions`. Correct placement matters — it determines which contributors see the issue, and the repository structure is itself a map of where different kinds of decisions are made.

3. Deliberation — GitHub issue thread Once an issue is open, substantive discussion moves from Discord into the issue thread itself. This is where the problem is examined rigorously: assumptions are challenged, alternatives are considered, implications for other systems are identified, and blocking objections are raised and addressed. The issue thread is the deliberation record. Contributors who were not part of the original Discord conversation can engage here on equal footing. The goal of this stage is not to reach a decision but to reach clarity — a shared understanding of the problem and the range of viable responses to it.

To ensure issues receive attention, contributors should watch the repositories relevant to their work. Watching a repository means opting in to GitHub notifications for all activity in that repo — new issues, comments, and pull requests. In the early project, when the community is small, all contributors are encouraged to watch all repositories. As the project grows, watching the repositories corresponding to your working group is the minimum expectation.

When a significant issue is opened, the contributor who opened it should also post a brief note in the corresponding Discord working channel — linking directly to the issue. This is not a formal requirement but an expected community norm. It closes the gap between where conversation begins and where formal work is recorded, and ensures that an issue does not sit unread simply because a contributor was not watching the right repository.

4. Proposal — GitHub issue or dedicated document When deliberation has produced sufficient clarity, a specific proposal is articulated. This may happen within the issue thread itself or as a separate document linked from it, depending on the complexity of the change. The proposal states precisely what is being proposed, why this approach is preferred over alternatives considered, what its implications are for other systems or processes, and what a successful outcome looks like. At this stage the proposal is open for final review. Contributors may express support, raise remaining concerns, or identify conditions that must be met before they can support it.

5. Consensus — GitHub issue thread Integral's governance does not operate by majority vote. A proposal is considered ratified when no contributor maintains an unresolved blocking objection. A blocking objection is not a general preference for a different approach — it is a specific, substantiated concern that the proposal causes a clear harm or violates a foundational principle, accompanied by a concrete path toward resolution. When all blocking objections have been addressed or withdrawn, consensus is established. The ratification is noted in the issue thread and the proposal moves forward.

6. Pull Request — GitHub Ratified proposals that require changes to code, specifications, or documentation are implemented through a pull request. The pull request references the originating issue, describes what has been changed and why, and is reviewed for technical correctness before merging. In the early project, merge authority to the main branch rests exclusively with the trustee. As the core team develops and trust is established, this authority may be selectively delegated.

7. Decision Record — `integral/decisions` Any change that affects the architecture, a foundational principle, an interface contract, or the governance of the project itself is recorded as a Decision Record in the `integral/decisions` repository. Decision Records are append-only and immutable once ratified. They document the problem addressed, the proposal adopted, the alternatives considered, the reasoning behind the decision, and the date of ratification. They are the project's institutional memory — the record that allows future contributors to understand not just what the system is, but why it is that way. The creation of a Decision Record will be automated.

Implementation is tracked through the originating issue, which remains open until the change is confirmed complete. If the outcome proves inadequate or produces unintended consequences, a new issue is opened and the process begins again.

A Note on Scale

In the early stages of the project, with a small contributor community, this process will often be lighter in practice than the steps above might suggest. A concern raised in Discord may move to a GitHub issue and reach consensus within days, with a pull request following shortly after. The formal structure exists not to slow things down but to ensure that nothing important is decided informally, that the reasoning behind decisions is preserved, and that the process scales cleanly as the community grows. The discipline of the workflow matters most when the project is large enough that not everyone knows what everyone else is doing — which is exactly when there will no longer be time to establish it from scratch.

Appendix A — Glossary of Core Terms

This glossary defines terms as they are used within the Integral project. Where a term has a common usage in software engineering, economics, or another field that differs from its Integral-specific meaning, that difference is noted.

Note on scope: This glossary covers terms used in the current version of the development guide. Some entries describe concepts belonging to the federation layer of Integral — multi-node coordination, scope detection, equivalence bands, and related mechanisms — which are not yet covered in the main body of this document. These entries are included here because the terms appear in the data schemas and white paper. They will be developed fully in later versions of the guide as the federation architecture is specified.

Access Cost — The number of Integral Time Credits required to permanently acquire a produced good or service. Access costs are computed from embodied labor, material consumption, and ecological impact as recorded in the COS production ledger and reflected in the OAD design. Access costs are not prices — they do not fluctuate with demand, they cannot be negotiated, and they are not set by any market mechanism. Short-term borrowing of shared-use items does not incur an access cost.

Append-Only Ledger — A data store in which records can be added but never modified or deleted. Corrections are recorded as new entries that reference the original, not as modifications to it. All five Integral systems maintain append-only ledgers as their authoritative record of what occurred. This design makes tampering detectable and institutional memory permanent.

Architectural Faithfulness — The property of a simplified or minimal implementation that preserves all foundational principles of the full system even though many modules and features are not yet built. An architecturally faithful implementation can be expanded into the full system without redesigning its foundations. Contrast with functional approximation, which may produce similar outputs through different means that cannot scale into the full architecture.

Assurance Artifact — A structured record that travels with a design, labor contribution, or material provision and allows the receiving party to independently verify its provenance, quality, and conformity to standards — without requiring trust in the sending party as an authority. Assurance artifacts are the mechanism by which Integral establishes trust through verifiability rather than reputation or central oversight.

Bill of Materials (BOM) — The complete list of materials required to produce a good according to a certified OAD design, including quantities, units, and ecological flags for each material.

Blocking Objection — In CDS deliberation, a concern raised by a participant that must be addressed before a decision can be finalized — it cannot be overridden by numerical majority. A blocking objection must be substantive and must be accompanied by a proposed resolution.

Builders — One of five contributor types. Developers, engineers, and systems architects who translate specifications into working software.

CDS (Collaborative Decision System) — The governance system of Integral. Responsible for receiving issues and proposals, structuring deliberation, synthesizing consensus, recording decisions, and dispatching decisions to other systems for implementation.

Certified Design — An OAD design that has passed ecological assessment, bill of materials review, and certification panel approval, and has been published with a version-locked status that COS can verify. No production can begin in Integral without a certified design reference.

COS (Cooperative Organization System) — The production and operations system of Integral. Responsible for translating certified designs into organized work, managing labor assignment and completion, tracking materials consumption, performing quality assurance, and recording all production activity.

Credit Balance — The current state of a participant's accumulated Integral Time Credits after issuance, decay, and extinguishment events. Credits are non-transferable, decay over time, and represent recognized contribution rather than accumulated wealth. Credits are extinguished — not redeemed or transferred — when a participant permanently acquires a produced good.

Cybernetic — In the context of Integral, a system that governs itself through feedback — continuously sensing its own state, comparing that state to its goals, and adjusting its behavior accordingly. The FRS-to-CDS feedback loop is the primary cybernetic mechanism of Integral.

Data Contract — A formal agreement between two systems about what information will be exchanged, in what format, and with what guarantees. Data contracts are defined in `integral/specifications` and are versioned independently of the systems they connect.

Decay (ITC) — The scheduled reduction of a participant's credit balance over time, designed to prevent indefinite accumulation and the emergence of a class of permanent credit-holders. Decay is a foundational principle of the ITC — it is architectural, not configurable.

Decision Packet — The structured output of a CDS decision specifying what action is required, which system is responsible, what parameters apply, and what FRS should monitor as a success indicator.

Decision Record (DR) — A formally ratified governance decision stored in the `integral/decisions` repository. Named `DR-NNN-brief-title.md`. Never edited after ratification. Contains the decision, context, alternatives considered, reasoning, and participants.

Deferred — A module or feature intentionally excluded from the current implementation scope because it is premature — requires capabilities that do not yet exist, requires accumulated operational data, or depends on later federation conditions. Deferred items are retained in planning documents and are not abandoned.

Diagnostic Finding — A structured output of FRS diagnostic analysis representing a named concern about the system's operational state. Has a type, severity, persistence, confidence, and references to supporting evidence.

Dispatch — The act of CDS sending a decision packet to the appropriate operational systems for implementation.

Ecological Flag — A simple classification — low, medium, or high — indicating the ecological impact level of a material or produced good. Assigned during OAD Module 3 assessment and inherited by COS production records and ITC access cost calculations.

Embodied Labor — The total weighted labor hours that went into producing a good, as recorded in the COS production ledger. One of the primary inputs to ITC access cost calculation.

Equivalence Band *(federation layer — see scope note above)* — A bounded, public range within which a contribution made in one node is interpreted by another, used in cross-node ITC recognition. Equivalence bands are calibrated to reflect real differences in ecological conditions, skill scarcity, and infrastructural context without creating exchange rates or arbitrage opportunities. Because equivalence is interpretive and bounded — not a conversion rate — ITCs remain non-transferable and non-monetary across the entire federation.

Explorers — One of five contributor types. People drawn to the project who are not yet sure where they fit, needing a reading sequence, low-stakes first action, and community to think alongside.

Extinguishment — The precise term for the act of spending Integral Time Credits to permanently acquire a produced good. Credits are extinguished — they cease to exist — rather than transferred to another party. This is what prevents accumulation, trade, speculation, and the conversion of contribution into lasting power. Note that short-term borrowing of shared-use items (tools, equipment, infrastructure) does not trigger extinguishment — only permanent acquisition does. This distinction is encoded in the `access_type` field on every access event (`permanent_acquisition` vs. `shared_use`). Contrast with "redemption," which implies exchange, and "transfer," which implies the credit moving from one holder to another. Extinguishment means termination.

Federation — The architecture by which independent Integral nodes connect, share designs, recognize labor contributions, coordinate production, and exchange diagnostic intelligence without requiring a central authority or shared infrastructure. Nodes are autonomous before, during, and after federation.

FRS (Feedback & Review System) — The adaptive monitoring system of Integral. Responsible for reading operational data from COS, ITC, OAD, and CDS; diagnosing patterns and concerns; producing recommendations for other systems; presenting a plain-language sensemaking summary to participants; and submitting this to CDS as governance input. In later multi-node development, FRS also supports cross-node signal aggregation and scope detection.

ITC (Integral Time Credits) — The contribution accounting system of Integral. Responsible for recording labor events, computing credit values using skill weighting, applying time decay, managing credit balances, computing access costs, recording access extinguishments, and maintaining an append-only ledger. Credits are non-transferable and decay over time.

Interface Cooperative (IC) — A COS-oriented Integral cooperative legally structured to interact with the existing market economy while operating internally under Integral principles. ICs engage in market-facing transactions (sales, contracts, procurement) and receive monetary revenue from external entities, but treat that revenue as an input to be converted into commons assets — not as an objective. Contributors inside an IC are recognized through ITC, not wages. The IC's defining function is asset conversion: tools, machines, land, and infrastructure acquired through market activity are transferred into Integral commons governance and shared across nodes through COS provisioning. ICs are a transition mechanism, designed to become less necessary as the commons matures.

Labor Event — The atomic unit of contribution accounting. A record produced every time a verified unit of work is completed within a COS production workflow. Carries participant identity, task reference, hours worked, skill tier, and ecological flag. Note that the white paper separates raw labor capture (`LaborEvent`) from weighted contribution recognition (`WeightedLaborRecord`); the devguide consolidates these for Phase 2 simplicity, but the conceptual boundary between them must be preserved in implementation.

Makers — One of five contributor types. Writers, designers, communicators, and documentarians who create materials that make the project legible and navigable.

Merged CDS *(federation layer — see scope note above)* — A temporary, problem-scoped CDS instance convened when a problem crosses node boundaries and no single node's CDS has sufficient jurisdiction. It draws participants from all affected nodes, runs a structured deliberation process, records the outcome in the relevant ledgers, and then dissolves. It is intended to prevent the emergence of a permanent federation-level governance body.

Minimum Viable Module Set — The subset of each system's full module specification that must be operational in the minimum viable system so the proof of concept can be tested with all five systems connected in a working feedback loop, with all foundational principles present even if many modules are deferred.

Node — The primary organizational unit of Integral. A cooperative community of any size that operates all five Integral systems in a real production and governance context. A node is autonomous: it makes its own decisions through CDS, maintains its own ledgers, and certifies its own designs.

Non-Transferability — The foundational ITC principle that credits cannot be moved from one participant to another under any circumstances. Enforced architecturally — a permanent field in the ITCAccount schema, not a configurable policy.

OAD (Open Access Design) — The shared knowledge and design system of Integral. Responsible for receiving design submissions, managing version control, conducting ecological assessment, running certification review, publishing certified designs to the commons, and maintaining design lineage history.

Practitioners — One of five contributor types. Cooperative organizers, mutual aid workers, and commons practitioners with real experience in non-market coordination.

Production Mandate — A CDS decision that authorizes COS to begin production of a specific good based on a certified OAD design. No production begins without a production mandate.

Proto-Node — An early-stage Integral community or organizational context exploring partial adoption of Integral logic and preparing for fuller real-world node operation. A proto-node may simulate some functions, test governance practices, or operate only part of the architecture while building toward a complete node.

Reciprocity Primitive *(federation layer — see scope note above)* — One of five atomic inter-node flow types in the Integral federation. The five primitives are: (1) Information and Knowledge Flows (OAD designs, decision records, FRS models, training materials — non-rivalrous, no reciprocity obligation incurred); (2) Capability and Capacity Flows (time-bounded access to machines, specialist services, logistics, emergency buffers); (3) Labor Mobility Flows (individuals contributing to a node other than their home node, recognized through ITC equivalence bands); (4) Material and Goods Flows (physical provisioning between nodes, constrained by ecology and need, coordinated through COS); (5) Assurance and Validation Flows (certification results, safety data, provenance proofs, reliability histories).

Recommendation — A structured, non-executive output of FRS Module 4 suggesting an action to a specific target system. Recommendations are not commands. They are inputs that the receiving system can accept, reject, modify, or escalate.

Scope Detection *(federation layer — see scope note above)* — The FRS-operated process by which cross-node conditions are assessed against threshold functions to determine whether a problem exceeds local regulatory capacity. Scope detection operates on aggregated NSS vectors. When thresholds are crossed, it triggers Coordination Envelope instantiation. Scope detection is automatic and continuous — it does not require authorization and does not accumulate authority.

Scope Mismatch *(federation layer — see scope note above)* — The condition that obtains when a problem's effects propagate beyond the regulatory boundary of a single node — when local regulation is insufficient not because of failure but because the problem space is genuinely cross-nodal. Scope mismatch is the trigger condition for Coordination Envelope instantiation.

Seam Point — A location in the codebase where two systems exchange data — the natural future boundary between services in a federated architecture. Seam points are designed cleanly from the beginning so the transition from a single-node monolith to a federated multi-service architecture is a deployment change, not a rewrite.

Signal Packet — The periodic structured summary produced by FRS Module 1 from COS, ITC, OAD, and CDS data. Contains labor summaries, materials summaries, ITC summaries, QA summaries, OAD design event summaries, and CDS governance indicators. The primary input to FRS diagnostic analysis and the eventual basis for the sensemaking artifact submitted to CDS.

Skill Tier — A classification of labor into broad categories reflecting relative complexity and scarcity. The canonical four-tier enum used throughout this document and all system schemas is: `low | medium | high | expert`. Each tier carries a weight multiplier set by CDS policy.

Sensemaking Artifact — The plain-language summary produced by FRS Module 5. The community's primary window into its own operational reality.

Syntegrity — A high-bandwidth human deliberation architecture developed by cybernetician Stafford Beer. Used as the escalation mechanism inside CDS Module 9 when standard facilitated deliberation cannot resolve a dispute involving cultural meaning, ethical tension, or irreducible value conflict. Syntegrity distributes influence evenly, routes insight through designed rotation, and surfaces coherence that algorithmic inference cannot derive. It does not replace CDS — it is the constitutional last resort within Module 9 that prevents deadlock or arbitrary override. Outputs re-enter the formal CDS pipeline through Module 7 recording and Module 8 dispatch.

Thinkers — One of five contributor types. Systems theorists, governance scholars, economists, and domain experts who engage with the architecture at the level of ideas.

Vertical Slice — A development approach in which a thin but complete path through the entire system is built before expanding the depth of any single component. Vertical slicing is the antidote to component-by-component development, which risks producing parts that cannot be integrated.

Access Center — A physical distribution point at which participants extinguish credits and collect goods. One of several named infrastructure types through which ITC access operates, alongside Tool Libraries (shared-use equipment pools, typically free to borrow without credit extinguishment), Shared-Use Pools, and Distribution Nodes.

Coordination Envelope (CE) *(federation layer — see scope note above)* — A temporary, bounded coordination context instantiated automatically when FRS detects that a cross-node problem cannot be resolved by local action alone. A CE defines which nodes are affected, what constraints are shared, what decisions may be made within its scope, and the conditions for its dissolution. It is not a governing body or standing institution. Instantiation requires no authorization — it is a system response, not a political act. Dissolution is equally automatic when FRS exit thresholds are met. No institutional residue remains after dissolution.

Node State Summary (NSS) *(federation layer — see scope note above)* — A compressed, bounded representation of a node's outward-facing effects, published periodically so that FRS can aggregate cross-node conditions without requiring nodes to expose internal governance decisions or participant-level data. Covers four state classes: ecological state, capacity state, dependency state, and risk state.

Viability Envelope *(federation layer — see scope note above)* — The operational boundaries within which a node can self-regulate without external intervention. FRS Module 3 models viability envelopes through scenario simulation, enabling the community to assess whether current trajectories remain within viable ranges before constraints become crises. Related to but distinct from a Coordination Envelope: a Viability Envelope describes a node's own stability range; a Coordination Envelope is the inter-node response mechanism when that stability is threatened by cross-node conditions.

Appendix B — Recommended Reading and Reference Systems

This appendix points contributors toward the foundational intellectual traditions that Integral draws on, and toward existing tools and frameworks that inform specific module designs. It is not exhaustive — the white paper's own references are more complete — but it provides entry points for contributors who want to understand the theoretical and practical foundations of what they are building.

Foundational Theory

Cybernetics and Management Cybernetics — The intellectual foundation of Integral's feedback architecture. Stafford Beer's *Brain of the Firm* and *The Heart of Enterprise* describe the Viable System Model, which directly informs how the five systems relate to each other and how the FRS-to-CDS feedback loop is structured. Norbert Wiener's *Cybernetics* and *The Human Use of Human Beings* provide the deeper theoretical grounding. W. Ross Ashby's *An Introduction to Cybernetics* is the most accessible formal introduction to the field.

Commons Governance — Elinor Ostrom's *Governing the Commons* is the essential empirical reference for understanding how real communities have managed shared resources sustainably — and where they have failed. Her design principles for robust commons institutions directly inform Integral's node governance architecture and the anti-coercion provisions of ITC Module 7.

Cooperative Economics and Mutual Aid — Peter Kropotkin's *Mutual Aid: A Factor of Evolution* provides the historical and evolutionary basis for cooperative organization. J.K. Gibson-Graham's *A Postcapitalist Politics* documents the diversity of real non-market economic practices that already exist. The history of the Mondragon cooperatives provides the most extensively studied large-scale cooperative economic case study.

Post-Market Economic Coordination — Pat Devine's *Democracy and Economic Planning* develops the concept of negotiated coordination as an alternative to both markets and central planning — directly relevant to CDS's governance architecture. Michael Albert and Robin Hahnel's work on participatory economics engages many of the same coordination problems Integral addresses. W. Paul Cockshott and Allin Cottrell's *Towards a New Socialism* addresses the computational feasibility of non-market coordination.

Ecological Economics — Herman Daly's *Beyond Growth* and *Steady-State Economics* articulate the foundational critique of growth-dependent economic systems. Kate Raworth's *Doughnut Economics* provides an accessible contemporary framing of the planetary boundaries concept that underlies FRS ecological monitoring.

Reference Systems and Tools

Decidim — An open-source participatory democracy platform used by cities and organizations for civic deliberation and participatory budgeting. Directly relevant to CDS Module 1 (Issue Capture) and Module 5 (Participatory Deliberation Workspace). Available at decidim.org.

Loomio — An open-source collaborative decision-making tool designed for cooperatives and community organizations. Relevant to CDS Module 5 and Module 6. Available at loomio.com.

Polis — An open-source tool for large-scale opinion mapping and consensus finding. Uses statistical clustering to identify areas of agreement and disagreement. Relevant to CDS Module 6 (Weighted Consensus Mechanism). Available at pol.is.

Kialo — A structured argument mapping platform that organizes deliberation as a tree of claims, supports, and objections. Relevant to CDS Module 2 (Issue Structuring) and Module 5.

Open Policy Agent (OPA) — An open-source policy engine that evaluates structured rules against structured data, returning pass/fail results with explanations. Directly relevant to CDS Module 4 (Norms & Constraint Checking). Available at openpolicyagent.org.

Git and Merkle Trees — The version control concepts underlying Git — particularly the use of cryptographic hashing to create tamper-evident history — are directly relevant to the append-only ledger design used across all five Integral systems and to the integrity fields present in the data schemas.

Holochain — A framework for building distributed peer-to-peer applications without a shared blockchain. Its agent-centric architecture aligns conceptually with Integral's emphasis on node autonomy and distributed coordination. Available at holochain.org. This is a reference technology and potential technical component, not a committed architectural choice.

Valueflows — An open vocabulary and protocol for describing flows of economic resources, work, and knowledge in cooperative and commons-based networks. Directly relevant to the COS production ledger and ITC contribution accounting data models. Available at valueflo.ws.

OpenLCA — An open-source lifecycle assessment tool among the most developed existing approaches to ecological accounting. Relevant to OAD Module 3's Material and Ecological Coefficient Engine. Available at openlca.org.

Community and Practice References

Platform Cooperativism — The movement to create cooperatively owned and governed digital platforms. The Platform Cooperativism Consortium documents existing examples and emerging practices.

Solidarity Economy — The broader ecosystem of cooperative, mutual aid, commons, and non-market economic organizations. The New Economy Coalition and Solidarity Economy Association document this ecosystem in North American context.

Time Banking — Existing time-based exchange systems — most notably those inspired by Edgar Cahn's work — provide direct practical reference for ITC design. The limitations of existing time banks are as instructive as their successes: understanding why they have not scaled, where contribution accounting creates perverse incentives, and what makes them sustainable or fragile in practice is essential background for ITC module development. TimeBanks USA and hOurworld are the primary North American networks.

Appendix C — The Building Blocks of Software: A Map for Non-Technical Contributors

This appendix is written for contributors who are new to programming and software architecture. Its purpose is not to teach you to code — that is a separate and much longer undertaking. Its purpose is to give you a map of the conceptual territory so that the technical language in this guide and in the white paper becomes legible, and so you can participate in architectural discussions without needing to treat technical decisions as a black box.

The core insight is this: a computer does exactly one thing. It moves and transforms data. Everything in software — every abstraction, every pattern, every framework, every architectural principle — is an answer to a single question: *how do we organize the movement and transformation of data so that it remains correct, understandable, and maintainable as the system grows?*

The hierarchy below runs from the most concrete and small (individual values stored in memory) to the most abstract and large (how entire systems relate to each other). Each level rests on the one below it. Understanding the levels helps you understand why decisions made at one level have consequences several levels up — and why a seemingly small change to a data field can require a governance decision involving the entire community.

Level 1 — Values

The most primitive thing in software is a value: a single piece of data occupying a known type. A number. A piece of text. A true-or-false. A point in time.

In Integral: `4.5` (a number of hours), `"technical"` (a skill tier label), `true` (a boolean — was this task verified?), `2025-03-07T14:32:00Z` (a timestamp).

Values alone mean nothing. `4.5` could be hours worked, a credit balance, an ecological impact score, or the result of a calculation. What gives a value meaning is the context it appears in — which is provided by the next level.

Level 2 — Fields and Records

A **field** is a named slot that holds a value. Naming a value creates meaning: `hours = 4.5` is now not just a number but a specific measurement of a specific thing.

A **record** (also called a struct, object, or row depending on the programming language and context) is a collection of named fields that together describe one coherent thing — one event, one participant, one decision, one material consumption.

The `LaborEvent` described in Section 4 is a record. It contains fields including `event_id`, `participant_id`, `hours`, `skill_tier`, `verification_status`, `timestamp`, and others. Together these fields describe one act of verified labor by one participant at one point in time. No single field tells you that — the record as a whole does.

Schemas are formal descriptions of records: what fields a record contains, what type of value each field holds, and which fields are required versus optional. When Section 4 of this guide says "this field must exist from day one even if empty," it is saying: this field belongs in the schema now, even though the module that populates it has not yet been built. The shape of the record must anticipate the full system even when the full system does not yet exist.

Level 3 — Collections

Records rarely exist in isolation. A **collection** is a set of related records stored together: all the labor events for a given period, all the credit balances in a node, all the pending governance proposals.

Collections have properties that matter architecturally. The most important in Integral is **append-only**: a collection to which records can be added but never modified or deleted. When something needs to be corrected, a new record is added that references the original and describes the correction. The original is never touched.

This is not a technical quirk. It is a principle. Append-only ledgers make tampering detectable — you cannot quietly change a historical record because any change would break the chain of references. They make institutional memory permanent — the history of what happened is preserved exactly as it happened, not as anyone later wished it had happened. Every system in Integral — CDS, OAD, ITC, COS, FRS — maintains append-only ledgers for exactly this reason.

Collections also have **indexes** — structures that make finding specific records fast without reading everything — and **ordering**, typically by timestamp. These are implementation details, but worth knowing exist.

Level 4 — Functions

A **function** is a named, reusable transformation: it takes some inputs, performs some operations on them, and returns an output. Functions are where logic lives.

`compute_credit_value(hours_verified, skill_tier, weight_table)` — this is a function signature. It declares: give me verified hours, a skill tier, and a weight table, and I will return a number of credits. The body of the function — the actual calculation — implements the ITC's weighting logic. Change how the function works and you change how the system values labor.

Functions are the atomic units of behavior. A module is, in large part, a collection of related functions that together implement a capability.

One property of functions worth understanding: **pure functions** produce the same output for the same input every time, with no side effects — they do not write to databases, send messages, or change anything outside themselves. **Impure functions** do have side effects — they write records, update balances, trigger events. In a system like Integral where data integrity is paramount, knowing which functions are pure and which have side effects matters. Pure functions are easy to test and reason about. Impure functions require more careful governance.

Level 5 — Modules

A **module** is a named, bounded unit of software that groups related functions and the data they operate on. It has a defined purpose, a defined set of things it is responsible for, and — critically — a defined boundary.

COS Module 4 (Cooperative Workflow Execution) is a module. It contains the functions that manage task lifecycle: creating tasks, assigning them, recording progress, triggering verification, recording completion. It knows about tasks, assignments, and status transitions. It does not touch credit balances — that is ITC's responsibility. It does not generate governance signals directly — that is FRS's responsibility. The boundary is the point.

Encapsulation is the principle that a module's internal workings are its own business. Other modules do not reach inside and manipulate its data directly. They ask it to do things through defined channels. This is what makes it possible to change how a module works internally — improving an algorithm, optimizing a query, restructuring internal data — without breaking everything that depends on it.

The white paper's full specification describes modules in detail: what each one does, what its inputs and outputs are, how it connects to other modules within the same system. Section 3 of this guide describes which modules belong in the minimum viable implementation and which are deferred until later development.

Level 6 — Interfaces

An **interface** is a formal contract describing how two modules or systems communicate. It specifies: what functions are available from the outside, what inputs each function accepts, what outputs it returns, and what errors it can produce.

```
get_labor_events(node_id, period_start, period_end) → List[LaborEvent]
```

This is an interface definition. ITC calls this function on COS to retrieve labor records for a given period. ITC does not know — and must not need to know — how COS stores labor events, what database it uses, or how it structures its internal queries.

This separation — called **loose coupling** — is what allows COS and ITC to be built by different contributors, maintained independently, and evolved at different rates without breaking each other.

Interfaces in Integral live in the `integral/specifications` repository and are the most carefully governed part of the codebase. A change to an interface potentially affects every system that depends on it.

Level 7 — Systems

A **system** is a collection of modules that together fulfill a major function of the architecture. In Integral, the five systems — CDS, OAD, ITC, COS, FRS — operate at this level.

Each system has its own modules, its own internal data, its own logic. Systems communicate with each other only through defined interfaces.

In the early implementation, all five systems may live inside a single codebase running on a single server — a **monolith**. Even in that environment the service boundaries must be respected. Code in COS must not directly manipulate ITC's data structures. It must go through the defined interface.

This discipline is what makes it possible to later separate the systems into independent services as the network grows and multiple nodes begin operating.

Level 8 — Architecture

Architecture is the set of decisions about how systems relate to each other, where authority sits, how data flows across the whole, and what constraints are inviolable.

In Integral, architectural decisions include: FRS can read from COS but never write to it; credits carry `transferable: false` as a permanent schema field; every system maintains an append-only ledger; OAD certification is required before COS production begins; participant IDs are structured as `node-id:participant-id` from the beginning.

Architecture is not a single diagram or specification. It is the consistent pattern of decisions across the entire codebase.

Architectural faithfulness means that a simplified early implementation preserves these architectural principles even when many modules are not yet built.

Level 9 — Infrastructure

Infrastructure is the layer beneath the application: the actual computers, networks, databases, and deployment machinery that run the software.

In the early implementation, Integral's infrastructure will likely be simple: a server running the application, a database storing the ledgers, and a web interface through which participants interact with the system.

As the network grows and multiple nodes operate independently, infrastructure may become more distributed: nodes communicating across networks, identity verification across systems, and protocols for validating claims made by one node to another.

How the levels map to the development guide

Development Guide Section	Primary Levels
Section 3 — Minimum Viable Modules	Level 5 (modules)
Section 4 — Critical Data Structures	Levels 2–3 (records and collections)
Section 4 — API Boundaries	Level 6 (interfaces)
Section 5 — Development Path	Levels 7–8 (systems and architecture)
Section 6 — Contributing to Integral	Organizational coordination across levels
White paper pseudocode	Levels 4–5 (functions and modules)
White paper system diagrams	Levels 7–8 (systems and architecture)

Why a small change can be a big decision

A schema change that appears small — for example adding a field to `LaborEvent` — may propagate through multiple levels of the system.

If the field affects the data returned by an interface, every system consuming that interface must adjust. If it affects how labor is interpreted, it may change ITC credit calculations or the signals FRS generates for CDS governance review.

This is why the guide emphasizes defining critical data structures before large amounts of code are written. Schema decisions ripple through the entire architecture.

A note on programming languages

The development guide does not prescribe a specific programming language. That decision is part of the project's early technology stack selection.

Regardless of language — Python, TypeScript, Go, Rust, or others — the conceptual levels described above remain the same. Records, collections, modules, interfaces, systems, and architecture exist in every language. Only the syntax changes.

What matters is the structure of the system being built, not the specific language used to implement it.

This appendix will expand over time as contributors identify concepts that need clearer explanation.

Version 0.1