Autonomous Agentic SDLC System — Master Plan (v0.1)

Last updated: 2025-08-21

0) Vision & Scope

Goal: Build an autonomous, agentic software team that takes a user request and executes the full SDLC—analysis → design → implementation → testing → security review → docs → deployment—then iterates on changes until the user is satisfied.

Success Criteria (initial): - Given a natural-language feature request, the system ships a working app or feature to a target environment with docs, tests, and changelog—without human edits—≥70% of attempts for scoped, greenfield web services. - Cycle time (request → deploy) ≤ 4 hours for small features; ≤ 48 hours for medium features. - Test coverage ≥ 80% statement for generated code; SLO: error budget < 1%/month. - All required artifacts exist per request: PRD, user stories, design/ADR, code, tests, runbook, user manual, release notes, wiki updates.

Non-goals (v0): Highly regulated domains (medical/aviation), deep legacy monolith refactors, and long-running migrations.

---

1) End-to-End Workflow (Happy Path)

1. **Intake**: User submits request (text + optional files). System creates a **Request Record** and a **Workspace**.
2. **Product Analysis** (PM Agent): Clarifies objectives, writes **PRD** and **User Stories**, negotiates scope with a Planner.
3. **Architecture & Design** (Architect Agent): Produces **High-level design**, **ADR(s)**, **Data model**, **API contracts (OpenAPI/GraphQL)**, **UML sketches**.
4. **Planning** (Planner/Orchestrator): Breaks work into **Issues/Tasks** with acceptance criteria; sets priority & dependencies.
5. **Implementation** (Dev Agents): Generate code in a branch, create **commits/PR**, self-review, and request Critic checks.
6. **Quality** (QA Agent): Generates tests (unit/integration/e2e), fuzz cases, security tests; runs in isolated **Execution Sandbox**.
7. **Security & Compliance** (SecOps Agent): SAST/DAST/dep scanning; license compliance; threat model.
8. **Docs** (Tech Writer Agent): Updates **User Manual**, **Runbook**, **Changelog**, **Wiki**.
9. **Merged & Deploy** (DevOps Agent): CI passes → auto-merge → deploy via IaC to **Staging**, then to **Prod** under policy gates.
10. **Observability & Validation** (SRE Agent): Smoke tests, canary, metrics/alerts; roll-forward/back as needed.
11. **Feedback Loop**: Collect telemetry & user feedback, open follow-up tasks; iteration restarts.

Fallback paths: HITL approval gates on risky changes; auto-revert and RCA if canary fails; auto-open bug with repro.

---

2) Agent Roster & Responsibilities

- **Intake Router**: Classifies request, detects missing info, triggers Q&A loop.
- **PM Agent**: Drafts PRD, user personas, success metrics, and user stories. Negotiates MVP vs Nice-to-have.
- **Architect Agent**: Chooses stack, defines boundaries, writes ADRs, designs APIs/data.
- **Planner/Orchestrator**: Task graph construction, tool routing, dependency mgmt, deadline/capacity modeling.
- **Dev Agents (N)**: Code generation/refactor, local builds, PR creation, code fixes.
- **Critic/Reviewer Agent**: Enforces style, linting, threat model checks, test adequacy, performance budgets.
- **QA Agent**: Test strategy, test-gen, flakiness triage, coverage gates, property-based testing.
- **SecOps Agent**: SAST/DAST, SBOM, supply-chain policies, secrets detection.
- **DevOps Agent**: CI/CD, IaC, env creation, rollbacks, artifacts registry.
- **Tech Writer Agent**: User manuals, API docs, runbooks, wiki.
- **SRE Agent**: SLOs, alerts, dashboards, incident runbooks.

Each agent exposes **Tools/Abilities** (see §3) and follows a **Message Protocol** (see §4).

---

3) Shared Services & Tooling

- **Memory & KB**: Vector DB for prior context; long-term project memory; retrieval policies.
- **Repo Service**: Git provider (GitHub) ops: create repo/branch, PRs, reviews, releases, tags.
- **Issue Tracker**: GitHub Issues for tasks/links to artifacts; labels: `type`, `risk`, `size`, `status`.
- **CI/CD**: GitHub Actions with reusable workflows; runners with containerized toolchains.
- **Execution Sandbox**: Ephemeral containers/VMs (e.g., Firecracker/Docker) per job; network policy; resource quotas.
- **Artifact Store**: Built assets, coverage reports, SBOMs, design images.
- **IaC**: Terraform + Pulumi (option A/B). Environments: `dev`, `staging`, `prod`.
- **Observability**: OpenTelemetry, Prometheus/Grafana, Loki, Jaeger; alert rules.
- **Security**: Trivy/Grype for deps, Semgrep for SAST, OPA/Conftest for policy, Gitleaks for secrets.
- **Documentation**: Docusaurus/ReadTheDocs + MkDocs; Mermaid for diagrams.

---

4) Orchestration & Message Protocol

- **Planner** constructs a Directed Acyclic Graph (DAG) of tasks with tool calls.
- **Message schema (JSON Lines)**:

```
{
  "role": "agent|tool|system",
  "agent": "pm|architect|dev|qa|secops|devops|writer|sre|critic",
  "task_id": "uuid",
  "parents": ["task_id"],
  "goal": "string",
  "inputs": {"...": "..."},
  "outputs": {"...": "..."},
  "tools": ["repo.create_branch", "ci.run", "kb.retrieve", "sandbox.exec"],
  "status": "queued|running|blocked|done|error",
  "artifacts": ["uri"]
}
```

- **Routing rules**:
- If missing acceptance criteria ⇒ return to PM.
- If coverage < gate ⇒ loop QA → Dev.
- If perf regression > threshold ⇒ Critic requests optimization.
- If security policy fail ⇒ SecOps blocks merge.

---

5) Artifact Templates (Canonical)

PRD.md

```
# Problem
# Goals / Non-goals
# Personas & Scenarios
# Requirements (Must/Should/Could)
# Success Metrics
# Risks & Assumptions
```

USER_STORIES.yaml

```
- id: US-####
  persona: "end-user"
  story: "As a <persona>, I want <capability> so that <outcome>."
  acceptance_criteria:
    - Given ... When ... Then ...
  priority: Must|Should|Could
  estimates:
    size: XS|S|M|L
    confidence: 0.6
```

ADR-YYYYMMDD-title.md

```
Context
Decision
Alternatives
Consequences
```

API/openapi.yaml — machine-checkable contracts.

TEST_PLAN.md

```
Scope
Test Pyramid (unit/integration/e2e)
Data & Fixtures
Coverage Targets
Non-functional (perf/security/usability)
```

RUNBOOK.md

```
Service Overview
SLO/SLA & Alerts
Dashboards
Common Failures & Fixes
Release & Rollback
```

USER_MANUAL.md for end users; **CHANGELOG.md**; **RELEASE_NOTES.md**.

---

6) Repository & Wiki Structure

```
repo/
  .github/workflows/
    ci.yml
    pr-quality.yml
    release.yml
  infra/
    terraform/
    k8s/
  services/
    webapp/
    api/
    worker/
  libs/
```

```
tests/
  unit/
  integration/
  e2e/
docs/
  prd/
  adr/
  api/
  manuals/
  runbooks/
tools/
  scripts/
.devcontainer/
CODEOWNERS
CONTRIBUTING.md
SECURITY.md
CHANGELOG.md
```

Wiki - /Process/SDLC-Policy - /HowTo/Local-Dev - /HowTo/Oncall-Runbook - /Architecture/Systems-Map

---

7) CI/CD Gates (Policy-as-Code)

- **Pre-commit**: formatting, lint.
- **PR gates**: build, unit tests, coverage ≥80%, SAST, license check, secrets scan, API contract tests.
- **Pre-deploy**: integration/e2e, migration dry-run, perf smoke, SBOM signed.
- **Post-deploy**: canary health, error rate < threshold for N minutes before full rollout.

OPA/Conftest rules block merges if gates fail. All gates produce artifacts.

---

8) Execution Sandbox Strategy

- Job per agent step with CPU/RAM caps.
- Network egress allowlist.
- File system snapshotting for reproducible builds.
- Escape hatches for HITL debugging with audit logs.

---

9) Safety, Governance, & HITL

- **Human checkpoints (configurable):**
- Approve PRD for net-new products.
- Approve ADRs that change data contracts.
- Approve production deployments above risk score X.
- **Risk scoring** from: LOC touched, critical files, secrets exposure, migration presence, blast radius.

- **Auditability**: immutable logs, signed commits, provenance (SLSA level target ≥ 3).

---

10) Tech Stack (suggested defaults)

- **LLM backbone**: pluggable (OpenAI, local, ensemble); toolformer/orchestrator component.
- **Runtime**: Python & TypeScript agents; LangGraph-style workflows.
- **Storage**: Postgres for state; Redis/RabbitMQ for queues; MinIO/S3 for artifacts; Vector DB (PGVector/Weaviate).
- **Frontend**: Next.js + Docusaurus for docs portal & request UI.
- **Infra**: Kubernetes or ECS; Terraform; GitHub Actions runners.

---

11) Milestones (8-week MVP)

Week 1-2: Foundations - Repo bootstrap, actions, Issue templates, policy gates skeleton. - Orchestrator MVP: Intake → PM → Architect → Planner sequence with stubs.

Week 3-4: Build & Test Loop - Dev Agent that can generate a minimal web service + tests, PR open. - QA Agent adds unit/integration tests; coverage gate enforced.

Week 5-6: Deploy & Docs - DevOps Agent provisions ephemeral staging via Terraform; release pipeline. - Tech Writer Agent generates User Manual + CHANGELOG per release.

Week 7: Security & Observability - SAST/DAST, SBOM, alerts + dashboards; incident runbook.

Week 8: Hardening & Pilot - Canary deployment, rollback automation, risk model; pilot 3 real features end-to-end.

---

12) Evaluation & Telemetry

- **Build quality**: pass rate, coverage, flakiness.
- **Delivery**: lead time, MTTR, deployment frequency.
- **User satisfaction**: thumbs-up rate on artifacts & deployed features.
- **Cost**: $/request, tokens/job, infra utilization.

---

13) Risks & Mitigations

- **Hallucinated requirements** → strict acceptance criteria, synthetic tests, contract-first APIs.
- **Security regressions** → mandatory gates, least privilege, secret scanners.
- **Flaky tests** → quarantine lanes, flake triage bot, retry budget.
- **Vendor lock-in** → pluggable LLM/provider interfaces.

14) Next Actions (Proposed)

1. Approve this plan (comment inline).
2. Decide initial **reference stack** (Python/TS + Flask/FastAPI + Postgres + Next.js?).
3. I will scaffold a **template monorepo** with workflows, policy gates, and all artifact templates.
4. Configure GitHub repo + Actions secrets; choose cloud (or local kind/K3d) for staging.
5. Pilot with a simple feature request: “Create a REST service to manage notes with auth and docs.”

15) Appendices

A. Issue Labels: `area/*`, `type/{feature,bug,ops,docs}`, `risk/{low,med,high}`, `size/{XS,S,M,L}`.

B. Example Policy (OPA)

```
package pr.gates
allow { input.coverage >= 0.8 }
deny[msg] { input.secrets_found > 0; msg := "secrets detected" }
```

C. Example GitHub Actions (outline)

```
name: CI
on: [pull_request]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
      - uses: actions/setup-python@v5
      - run: make ci
```