Creative Automation PoC

Outside-In Development for Enterprise AI

Forward Deployed Engineer Interview

Angelo Cisneros

2025-10-14

The Challenge: Build a PoC in days, not months

The Approach: Lean-Clean methodology + Pragmatic Clean Architecture

Roadmap: 5-Day PoC → Production Evolution

Phase 1: Foundation (Day 1) ✅

- Core entities & domain model
- Campaign brief schema (YAML)
- Fake adapters for all external services
- Acceptance tests with fakes

Phase 2: Orchestration ✅

- Orchestrator workflow implementation
- CLI + Streamlit drivers
- Multi-stakeholder workshop validation
- Iterate on fakes with stakeholders

Phase 4: Production Ready (Day 5) 📅

- E2E smoke tests
- MinIO → S3 storage migration
- Monitoring & logging
- Stakeholder demo

Phase 5: Adv. Features (Week 2-3) 📅

- Weaviate vector search (asset reuse)
- Brand compliance validation
- Legal content filtering
- A/B test framework

Phase 6: Agentic System (Week 4) 📅

The Enterprise PoC Problem

Traditional Approach:

```
Week 1-2:   Creative team writes requirements
Week 3-4:   Legal reviews, finds issues
Week 5-8:   Engineering builds, discovers Ad Ops needs weren't captured
Week 9-10:  IT flags integration problems
Week 11-12: Rework and compromises

Cost: $5,000+ in API bills | Time: 6-8 weeks | Result: Frustrated stakeholders
```

Lean-Clean Approach:

```
Day 1: 90-min workshop → executable test captures ALL requirements
Day 2-3: Implement orchestrator with fakes → stakeholders see it working
Day 4-5: Implement real adapters in parallel → production-ready PoC

Cost: ~$200 in API testing | Time: 5-6 days | Result: Aligned stakeholders
```

The Secret: Write realistic, executable tests **with stakeholders present** before

Multi-Stakeholder Workshop → Executable Spec

The 90-Minute Workshop:

Who's in the Room:

- Creative Lead | Legal/Compliance
- Ad Operations
- IT/DevOps

What They Define:

- Success criteria (as assertions)
- Edge case priorities
- SLA requirements
- Integration points

What You Deliver:

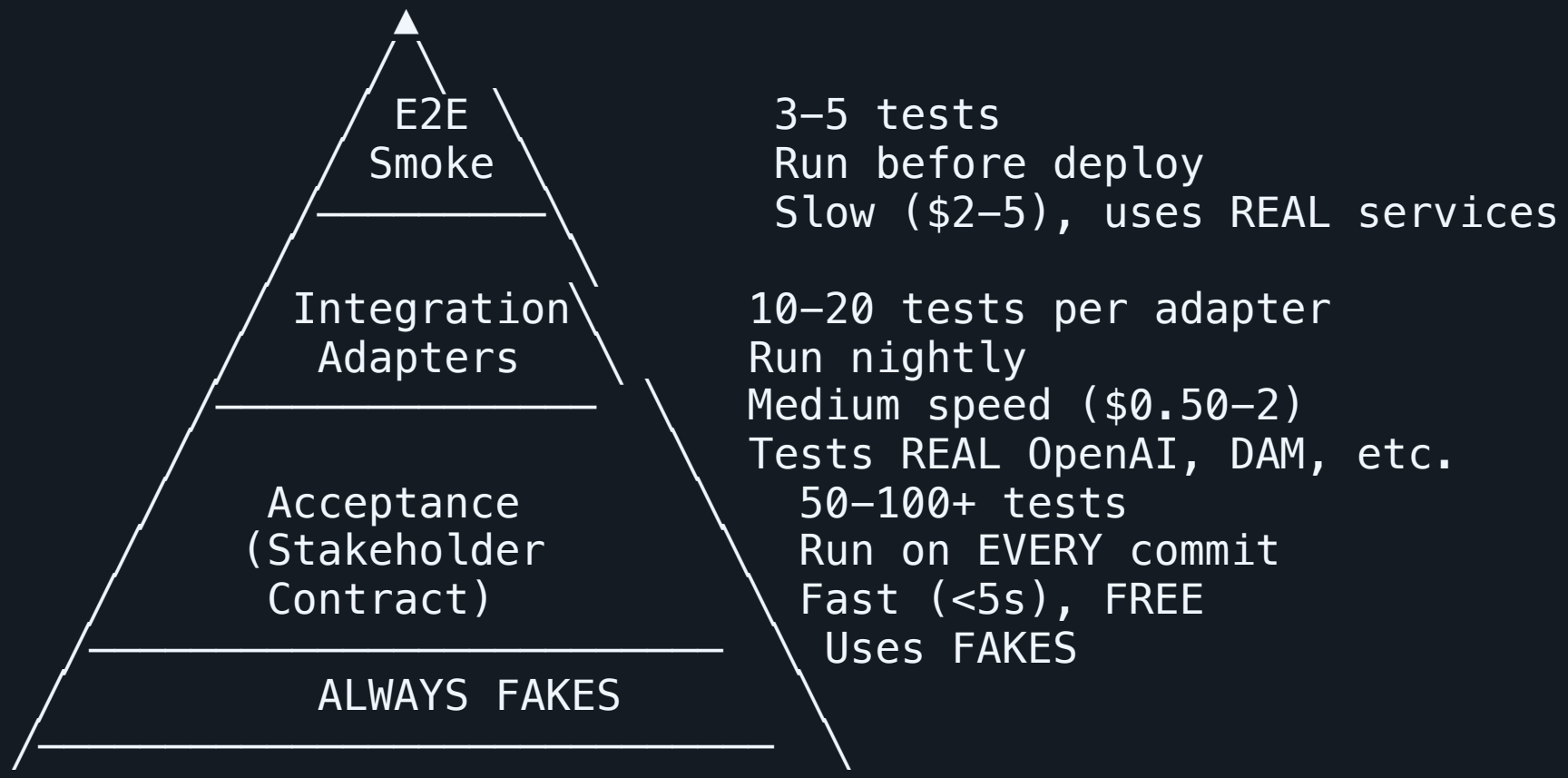
- Acceptance test (with fakes)
- All stakeholders see it run
- Iterate in workshop
- Sign-off before spending

Example Output:

```
async def test_campaign_generation():
    # Creative Lead requirement
    assert brand_compliant

    # Legal requirement
    assert content_filtered
```

The Testing Pyramid



3–5 tests
Run before deploy
Slow ($2–5), uses REAL services

10–20 tests per adapter
Run nightly
Medium speed ($0.50–2)
Tests REAL OpenAI, DAM, etc.

50–100+ tests
Run on EVERY commit
Fast (<5s), FREE
Uses FAKES

Layer 1 (Acceptance): ALWAYS uses fakes - this is your stakeholder contract

Layer 2 (Integration): Tests REAL adapters (OpenAI, DAM) in isolation

Layer 3 (E2E): Minimal smoke tests proving everything connects

System Architecture: Pragmatic Clean Architecture

```
Drivers Layer (ALWAYS PRESENT from Day 1)
──────────────────────────────────────────

CLI (Typer):          Streamlit UI:
- Automation          - Creative Lead: Visual review
- CI/CD integration   - Legal: Approve/reject
- Batch processing    - Ad Ops: Performance metrics
                      - IT: System monitoring

FastAPI (When Needed):
- Enterprise integrations
- External webhooks

          │ Calls orchestrators
          ▼
Orchestrator Layer
- Campaign workflow coordination
- Multi-market batch processing
- Event emission (campaign_generated, ab_test_assigned)

          │
          ▼
Use Case Layer (Business Logic)
- Brand validation rules
- Content filtering (Legal)
- Cultural adaptation
- Regulatory compliance per market

          │
      ┌───┴───┐
      ▼       ▼
Adapters      Infrastructure
(External)    (Persistence)

- OpenAI      - Campaign repo
- Claude        (Postgres)
- DAM         - Creative repo
- Analytics     (MongoDB)
```

Dependency Inversion + Adapter Pattern

The Dependency Rule:

```
User Request (CLI/UI)
    ↓
Drivers
    ↓
Orchestrators
    ↓
Use Cases
    ↓
Entities ←─────────┐
    ↑              │
    │              │
    │ Uses via     │ Implements
    │ Interface    │ Interface
    │              │
Protocols  →  Concrete Adapters
```

Dependencies point INWARD

- Use Cases depend on interfaces
- Adapters implement interfaces
- Swap implementations without changing Use Cases

Adapter Substitution:

```python
# Use Case depends on interface
class GenerateCampaign:
    def __init__(
        self,
        ai_adapter: IAIAdapter  # Interface!
    ):
        self.ai = ai_adapter

# Fake for workshop
class FakeAIAdapter(IAIAdapter):
    def generate_image(self, prompt):
        return placeholder_image()

# Real for production
class OpenAIAdapter(IAIAdapter):
    def generate_image(self, prompt):
        return openai.images.generate(...)

# Claude alternative
class ClaudeAdapter(IAIAdapter):
    def generate_image(self, prompt):
        return anthropic.messages.create(...)
```

Swap at runtime via config

Fakes: The Key to Velocity

Type 1: Fake Adapters

(External Services)

```
# Business stakeholders see these
class FakeImageGenAdapter:
    """Demo without API keys"""
    def generate_image(self, prompt):
        return self._placeholder_image()

class FakeDAMIntegration:
    """Simulate DAM upload"""
    def upload(self, asset):
        self.uploaded_assets.append(asset)
        return fake_dam_id()

class FakeEventTracker:
    """Capture events for assertions"""
    def track(self, event):
        self.events.append(event)
```

Use in: Acceptance tests

Purpose: Fast feedback, no API costs

Type 2: In-Memory Repositories

(Persistence)

```
# DevOps/IT stakeholders see these
class InMemoryCampaignRepo:
    """No database needed"""
    def __init__(self):
        self.campaigns = {}

    def save(self, campaign):
        self.campaigns[campaign.id] = campaign

    def get(self, campaign_id):
        return self.campaigns.get(campaign_id)

class PostgresCampaignRepo:
    """Production persistence"""
    def save(self, campaign):
        self.db.execute(
            "INSERT INTO campaigns..."
        )
```

Use in: Integration tests

Purpose: Isolate persistence logic

Critical Insight: Fakes live in production code (not test utils) so stakeholders can run them in workshops.

AI-Driven Monitoring & Alerts

Agent Responsibilities:

1. **Monitor** incoming campaign briefs
2. **Trigger** automated generation tasks
3. **Track** creative variant count/diversity
4. **Flag** missing/insufficient assets
5. **Alert** stakeholders with contextual info

Alert Types:

- ⚠️ Missing assets (< 3 variants)
- ⚠️ Generation failures (API errors)
- ⚠️ Validation failures

Model Context Protocol:

```
context = {
    "brief_id": "holiday-2025-01",
    "issue": "insufficient_variants",
    "expected": 12,
    "actual": 8,
    "missing": ["lavender-soap/es-US/9x16"],
    "reason": "OpenAI API rate limit"
}

# LLM generates stakeholder message:
"Holiday Gift campaign delayed.
Generated 8 of 12 assets. Missing
4 Spanish story-format images due
to API rate limits. Resolution: 2hrs.
Recommend: Approve English assets
now, Spanish separately."
```

Outcome: Structured data → LLM → Human-readable stakeholder communication

Key Design Decisions

Decision 1: Pragmatic Clean Architecture (not just scripts)

- ✅ **Pro:** Testable, maintainable, production evolution path
- ❌ **Con:** More upfront structure than quick script
- **Why:** Interview context + real-world PoCs need to scale

Decision 2: Dual AI Implementation (Fake + Real from Day 1)

- ✅ **Pro:** Demo without API keys, workshop velocity
- ❌ **Con:** More code to maintain (2x adapters)
- **Why:** Stakeholder alignment before spending

Decision 3: CLI + UI Drivers from Day 1 (not just CLI)

- ✅ **Pro:** Creative Lead needs visual feedback, not logs
- ❌ **Con:** More complex than CLI-only PoC

Trade-offs & Constraints

Decision 4: YAML for Campaign Briefs (not JSON or API)

- ✅ **Pro:** Human-readable, comment-friendly, version control
- ✅ **Pro:** Stakeholders can hand-craft without tooling
- ❌ **Con:** Requires parser, not as ubiquitous as JSON
- **Why:** Lean-Clean principle - optimize for human readability

Decision 5: Campaign-Level Localization

- ✅ **Pro:** "One campaign, many locales" natural model
- ✅ **Pro:** Simpler implementation for 6-8 hour timebox
- ❌ **Con:** Product names not localized (assumes English)

Key Meta-Decision: Optimize for **interview context** (demonstrate skill + thinking) while maintaining **production viability** (not throwaway code).

Questions for Discussion

Architecture:

- Thoughts on Pragmatic CA vs. simpler patterns for PoCs?
- Experience with adapter pattern for AI service substitution?

Stakeholder Alignment:

- Experience with multi-stakeholder workshops?
- How would you handle conflicting requirements?

Forward Deployment:

- What challenges do you see in deploying this to customers?
- How would you adapt this for different customer contexts?

Thank You

Repository: GitHub link

Let's discuss:

- Deep dive on any architectural decisions
- Testing strategy refinements
- Real-world deployment considerations
- Forward deployment scenarios

```
  __                    ______
 / /  ___ ___ ____     / ___/ /__ ___ ____
/ /  / _ \/ _ `/ _ \___/ /__/ / _ \/ _ `/ _ \
/ /__/  __/ /_/ / / / /___/ /__/ /  __/ /_/ / / /
/____/\___/\__,_/_/ /_/    \___/_/\___/\__,_/_/ /_/
```

Lean-Clean Methodology:

The Secret Sauce: Outside-In with All Stakeholders