HIGHTOWER.DIGITAL® · CREATE EXCELLENCE!!™

IT Requirements Solutions — x5 Use Case Report

RESEARCH REPORT · VERSION R1.A

Author: LW Atkinson (CEO/CTO), HighTower Digital Inc.

Date: 06/25/2026 · **Classification:** HDI Confidential

Methods: Agile Scrum + Lean + XP · **Product:** IT Requirements As-A-Service (MSP SLA #1–5)

IT Requirements Solutions — x5 Use Case Report (R1.A)

HDI Confidential Document · LW Atkinson (CEO/CTO), HighTower Digital Inc. · *IT Requirements Solutions x5 Use Case Report R1a*

Author: LW Atkinson (CEO/CTO) · **Date:** 06/25/2026 · **Version:** R1.A

Product Context: HighTower.Digital — *IT Requirements As-A-Service* (MSP SLA #1–5: On-Demand, On-Site, Remote). *"Zero translation loss. Sprint-ready specs."*

Executive Summary

HighTower.Digital Inc. delivers **IT Requirements As-A-Service** under guaranteed SLA retainers (Advisory `$4,500/mo`, Partner `$8,500/mo`, Emergency Owner `$15,000/mo`) with a 15-minute emergency response and a single promise: **zero translation loss between executive strategy and developer reality.** This research report establishes the current industry standard for the **five IT Requirements service types** HighTower.Digital provides to remote and on-site enterprise customers, and converts each into a repeatable, sprint-ready specification practice grounded in **Agile Scrum + Lean + Extreme Programming (XP)**.

The report is structured around the international requirements-engineering baseline defined by **ISO/IEC/IEEE 29148:2018** (which superseded IEEE 830-1998), the **IIBA BABOK® Guide v3** business-analysis body of knowledge, the **2020 Scrum Guide**, and the prioritization disciplines of **MoSCoW**, **WSJF (Weighted Shortest Job First)**, and **RICE**. Each of the five service types is documented as a distinct specification deliverable — Design Spec, Development Spec, Product Spec, Technical Spec, and Domain/Brand Spec — with worked representative examples drawn from a `React/TypeScript` front end, a `Node.js` middleware tier, and a `Python` back end.

> **How to read this report.** Sections 1–5 mirror the five service types. Each follows the same skeleton: a validated design document, layered feature lists with worked **Feature Cards**, and a Technical Requirements block (business cases, edge cases, use cases, a Technical Requirements Card, and a prioritized index). The Appendices consolidate the methods, standards, and a full reference list.

Table of Contents

IT Requirements Index (Service Catalog)

The five primary **IT Requirements As-A-Service** types HighTower.Digital provides, mapped to their owner role, audience, and the specification document each produces.

#	Service Type	Owner Role / Audience	Specification Produced	Primary Standard
1	**Customer Discovery**	Team Member @ Dev, Client	Design Spec	ISO 29148 StRS / Lean Discovery
2	**Voice of the Customer**	Team Member @ Dev, Client	Development Spec	BABOK Elicitation / Kano / QFD

#	Service Type	Owner Role / Audience	Specification Produced	Primary Standard
3	**Product Owner**	Team Leader @ Dev, Exec, Client	Product Spec	Scrum Guide 2020
4	**Technical Owner**	Technical Leader @ Enterprise	Technical Spec	ISO 29148 SRS / NFR / ADR
5	**Domain Owner**	Brand Management @ Enterprise	Domain / Brand Spec	DDD / Domain Governance

01

Customer Discovery

Design Spec

02

Voice of Customer

Development Spec

03

Product Owner

Product Spec

04

Technical Owner

Technical Spec

05

Domain Owner

Brand Spec

1 Customer Discovery Requirements (Design Spec Document)

Customer Discovery is the front end of requirements engineering. Following Steve Blank's Customer Development model and the Lean Startup loop, the goal is to **"get out of the building"** and validate the problem before committing engineering capacity. The deliverable is a **Design Spec** that converts unverified assumptions into evidence-backed problem, solution, and outcome statements — the StRS (Stakeholder Requirements Specification) layer of ISO/IEC/IEEE 29148.

1.1 Customer Discovery Use Case Statement

A discovery use-case statement frames the engagement using the **Jobs-To-Be-Done (JTBD)** lens — *"When [situation], I want to [motivation], so I can [expected outcome]"* — combined with the ISO 29148 problem framing. Below is the worked HighTower.Digital reference case used throughout Section 1: an enterprise client whose business requirements never survive the handoff to developers.

1.1.1 PROBLEM DESCRIPTION

Enterprise stakeholders express needs in business language; by the time those needs reach the development team they have been re-interpreted three or four times, producing rework, missed sprints, and "that's not what we asked for" at review. The measurable symptom: **38% of committed sprint stories are reopened** for clarification, and average lead time from request to merged spec is **9 business days**.

1.1.2 SOLUTION REQUIRED

A continuous, SLA-backed requirements triage service that ingests raw business needs and emits **developer-ready specifications** — API schemas, data models, acceptance criteria, and edge-case definitions — with traceability back to the originating stakeholder need, all within the client's existing sprint cadence.

1.1.3 BENEFITS NEEDED

- Reduce sprint story reopen rate from 38% to **< 10%**.
- Cut request-to-spec lead time from 9 days to **≤ 1 business day** (Partner tier) / **15 minutes acknowledgment** (Emergency tier).
- Achieve **100% developer-ready** specs (no open interpretation at sprint planning).

1.1.4 SYS/ECO CONSTRAINTS

- Must integrate with the client's existing tooling (Jira Cloud / Azure DevOps) — no rip-and-replace.

- Must operate **remotely and on-site** under the same SLA.
- Must comply with the client's data residency and security policy (e.g., ISO 27001, SOC 2).
- Real-time data systems impose a hard latency budget; specs must carry NFR latency targets.

1.1.5 REQUIRED OUTCOMES

A signed-off **Design Spec** containing: validated problem statement, prioritized feature/function lists, and a Technical Requirements index ready to feed Section 2 (Voice of the Customer).

> **Standard applied.** ISO/IEC/IEEE 29148 requires every requirement to be *necessary, unambiguous, complete, singular, feasible, verifiable, correct, and conforming*. The discovery statement above is written so each outcome is measurable and therefore *verifiable*.

1.2 Survey / Discovery Process — QA + Results

Discovery is run as five timestamped question gates, each built on the **5 W's + H** (Who, What, When, Where, Why, How). This mirrors the Lean *Build–Measure–Learn* validation cadence and "The Mom Test" principle of asking about past behavior rather than future hypotheticals.

1.2.1 `2026-06-01 09:00 EST` — PRE-VALIDATION QA

W	Question
Who	Who currently writes the requirement, and who consumes it downstream?
What	What artifact is produced today, and where does it break?
When	When in the sprint does the breakdown surface?
Where	Where (which tool/channel) does the requirement live?
Why	Why does the current handoff lose fidelity?
How	How is success measured today (if at all)?

1.2.2 `2026-06-04 10:30 EST` — VALIDATION QA

Confirms the problem with quantified evidence (reopen rate, lead time pulled from Jira). Each answer is logged against the originating stakeholder for traceability.

1.2.3 `2026-06-09 14:00 EST` — POST-VALIDATION QA

Validates the proposed solution shape with a low-fidelity spec sample; measures stakeholder comprehension ("can your developer build from this with zero questions?").

1.2.4 `2026-06-16 11:00 EST` — POST-VALIDATION-REVIEW 1 QA

Reviews the first real spec produced under the service against the success metrics; captures variance.

1.2.5 `2026-06-23 11:00 EST` — POST-VALIDATION-REVIEW 2 QA

Confirms sustained improvement across a full sprint; signs off the Design Spec for promotion to Development Spec.

> **Result of worked case:** reopen rate fell to **11%** after Review 1 and **7%** after Review 2; request-to-spec lead time dropped to **0.8 days** on the Partner tier — clearing the §1.1.3 targets.

1.3 Product / Service Features List

The discovery output is decomposed into candidate **features** (capabilities a user perceives). Representative set for the reference case:

- **F-01 Requirement Intake Portal** — single channel to submit raw business needs.
- **F-02 SLA Timer & Acknowledgment** — 15-min response clock with audit trail.
- **F-03 Spec Generator** — converts intake into a structured, developer-ready spec.
- **F-04 Traceability Matrix** — links each requirement to stakeholder, story, and test.
- **F-05 Tool Sync** — bidirectional Jira/Azure DevOps integration.

1.4 Product / Service Functional List

Functions are the system behaviors that realize the features (ISO 29148 functional requirements):

ID	Function	Realizes
FN-01	Submit requirement with attachments and stakeholder tag	F-01
FN-02	Start/stop SLA timer; emit acknowledgment within 15 min	F-02

ID	Function	Realizes
FN-03	Parse intake into INVEST-compliant user stories	F-03
FN-04	Generate Gherkin acceptance criteria	F-03
FN-05	Create/maintain Requirements Traceability Matrix (RTM)	F-04
FN-06	Push approved story to Jira backlog via REST API	F-05

1.5 Technical Requirements

1.5.1 BUSINESS CASES

The service must demonstrably reduce rework cost. Business case BC-01: *"Reducing sprint reopen rate from 38% to <10% recovers ~3.4 developer-days per sprint per team."* This is the ROI hook that justifies the retainer.

1.5.2 EDGE CASES

- Intake submitted with **no measurable outcome** → system flags as "unverifiable" and blocks promotion (enforces ISO 29148 *verifiable* criterion).
- Stakeholder retracts/changes the need mid-sprint → RTM versioning records the change and impacted stories.
- SLA timer crosses a non-business-hours boundary (Advisory tier) → clock pauses per contract.

1.5.3 USE CASES

UC-01 Submit & Triage: Client analyst submits a need → SLA timer starts → HighTower architect acknowledges within 15 min → triages into discovery gate 1.2.1.

1.5.4 'TECHNICAL REQUIREMENTS' CARD

TECH REQ CARD · TR-CD-01

Title: Unverifiable-requirement guardrail **Type:** Functional + Quality (Verifiability) **Source:** §1.1.5, §1.5.2 **Statement:** *The system shall reject promotion of any requirement that lacks a measurable, testable acceptance criterion.* **Acceptance (Gherkin):**

```
Feature: Verifiability guardrail
  Scenario: Block unverifiable requirement
    Given an intake item with no measurable acceptance criterion
    When an analyst attempts to promote it to the backlog
    Then the system rejects promotion
    And displays "Add a measurable acceptance criterion to continue"
```

Priority: MUST · **Verification:** Automated test + review gate

1.5.5 TECH REQUIREMENTS INDEX + PRIORITY

ID	Requirement	Priority	Method
TR-CD-01	Unverifiable-requirement guardrail	MUST	Scrum/XP test-first
TR-CD-02	15-min SLA acknowledgment	MUST	Lean flow
TR-CD-03	RTM auto-generation	SHOULD	Scrum
TR-CD-04	Jira bidirectional sync	SHOULD	XP CI
TR-CD-05	Discovery analytics dashboard	COULD	Lean metrics

2 Voice of the Customer (Development Spec Document)

Voice of the Customer (VOC) translates validated discovery into a **Development Spec**. Using BABOK elicitation, the **Kano model** (must-be, performance, delighter), and **QFD**

(Quality Function Deployment) to weight customer attributes against engineering characteristics, VOC produces the story backlog and the layered feature cards developers build from. Stories follow the **INVEST** criteria (Independent, Negotiable, Valuable, Estimable, Small, Testable) with Gherkin acceptance criteria.

2.1 Validated Design Document

2.1.1 STORY INDEX

Story	Title	Layer	Kano	Priority
ST-01	Submit a requirement	Frontend	Must-be	MUST
ST-02	See live SLA countdown	Frontend	Performance	SHOULD
ST-03	Route intake to architect	Middleware	Must-be	MUST
ST-04	Generate spec from intake	Backend	Performance	MUST
ST-05	Suggest acceptance criteria (AI)	Backend	Delighter	COULD

2.1.2 STORY BACKLOG

The backlog is ordered by **WSJF** (Cost of Delay ÷ Job Size). ST-03 and ST-04 carry the highest Cost of Delay because the spec pipeline is blocked without them; ST-05 is a delighter deferred to a later sprint.

2.1.3 STORY LIST (FEATURES / FUNCTIONS)

Each story maps to a Feature (§1.3) and Functions (§1.4). Example — **ST-01** realizes **F-01** via **FN-01**:

> *As a* client analyst, *I want* to submit a requirement with attachments and a stakeholder tag, *so that* it enters the SLA pipeline immediately. **AC:** Given a completed intake form, when I submit, then an item is created, the SLA timer starts, and I receive a ticket ID within 2 seconds.

2.2 Frontend Feature List

Stack: **React 18 + TypeScript**. Representative worked Feature Card below (ST-01); remaining frontend features (live SLA countdown, RTM viewer) follow the identical card template.

2.2.1 FEATURE CARD — REQUIREMENT INTAKE FORM

FRONTEND · FC-FE-01

Feature: Requirement Intake Form (realizes F-01 / ST-01)

(1) Header code

```
// IntakeForm.tsx — header & types
import { useState } from "react";
import { submitRequirement } from "../api/intake";

interface IntakePayload {
  title: string;
  description: string;
  stakeholder: string;
  attachments: File[];
}
```

(2) Function code

```
export function IntakeForm() {
  const [payload, setPayload] = useState<IntakePayload>({
    title: "", description: "", stakeholder: "", attachments: [],
  });
  const [ticketId, setTicketId] = useState<string | null>(null);

  async function handleSubmit(e: React.FormEvent) {
    e.preventDefault();
    if (!payload.title || !payload.description) return; // FN-01 guard
    const res = await submitRequirement(payload);       // starts SLA timer
    setTicketId(res.ticketId);                           // AC: <2s ticket id
  }
  return (
    <form onSubmit={handleSubmit} aria-label="Requirement intake">
      <input value={payload.title}
             onChange={e => setPayload({ ...payload, title: e.target.value })}
             placeholder="Requirement title" required />
      <textarea value={payload.description}
             onChange={e => setPayload({ ...payload, description: e.target.value })}
             placeholder="Describe the business need" required />
      <button type="submit">Submit to SLA pipeline</button>
      {ticketId && <p role="status">Ticket {ticketId} created.</p>}
    </form>
  );
}
```

(3) Footer code

```
// index.ts — barrel export
export { IntakeForm } from "./IntakeForm";
export type { IntakePayload } from "./IntakeForm";
```

(4) Use Case Example — A client analyst opens the portal, fills title + description, tags the VP of Product as stakeholder, and submits; a ticket ID returns instantly and the SLA timer begins.

(5) Step-by-Step Example 1. Navigate to `/intake`. 2. Enter title and description (client-side validation enforces non-empty — FN-01 guard). 3. Tag stakeholder; attach optional files. 4. Click **Submit** → `submitRequirement()` POSTs to middleware. 5. Receive ticket ID `< 2s`; status region announces it for screen readers (usability NFR).

2.3 Middleware Features List

Stack: **Node.js + Express/TypeScript**. The middleware authenticates, starts the SLA timer, and routes intake to an available architect.

2.3.1 FEATURE CARD — INTAKE ROUTER & SLA TIMER

MIDDLEWARE · FC-MW-01

Feature: Intake Router + SLA Timer (realizes F-02 / ST-03)

(1) Header code

```
// intakeRouter.ts — header
import { Router, Request, Response } from "express";
import { startSlaTimer } from "../sla/timer";
import { enqueueForArchitect } from "../queue/architectQueue";
export const intakeRouter = Router();
```

(2) Function code

```
intakeRouter.post("/intake", async (req: Request, res: Response) => {
  const { title, description, stakeholder } = req.body;
  if (!title || !description) {
    return res.status(422).json({ error: "title and description required" });
  }
  const ticketId = `HD-${Date.now().toString(36).toUpperCase()}`;
  await startSlaTimer(ticketId, req.user.slaTier);     // FN-02: 15-min clock
  await enqueueForArchitect(ticketId, { title, description, stakeholder });
  return res.status(201).json({ ticketId });           // AC: <2s
});
```

(3) Footer code

```
// app.ts — mount
import { intakeRouter } from "./routes/intakeRouter";
app.use("/api", intakeRouter);
export default app;
```

(4) Use Case Example — The POST from FC-FE-01 hits `/api/intake`; the router validates, mints a ticket ID, starts the tier-aware SLA timer, and enqueues the item for the next available architect.

(5) Step-by-Step Example 1. Receive POST `/api/intake`. 2. Validate payload (mirror of frontend guard — defense in depth). 3. Generate ticket ID; persist intake. 4. `startSlaTimer(ticketId, tier)` — 15 min (Emergency), 1 h (Partner), 4 h (Advisory). 5. `enqueueForArchitect()` and return `201` with ticket ID.

2.4 Backend Feature List

Stack: **Python (FastAPI)**. The backend turns triaged intake into INVEST-compliant stories with Gherkin acceptance criteria and persists the RTM.

2.4.1 FEATURE CARD — SPEC GENERATOR

BACKEND · FC-BE-01

Feature: Spec Generator (realizes F-03 / ST-04)

(1) Header code

```python
# spec_generator.py — header
from dataclasses import dataclass
from typing import List

@dataclass
class UserStory:
    story_id: str
    role: str
    want: str
    benefit: str
    acceptance: List[str]   # Gherkin lines
```

(2) Function code

```python
def generate_story(intake: dict) -> UserStory:
    """Convert triaged intake into an INVEST-compliant story (FN-03/FN-04)."""
    role = intake.get("stakeholder_role", "user")
    want = intake["description"].strip()
    benefit = intake.get("outcome", "achieve the stated business outcome")
    acceptance = [
        "Given a valid request",
        f"When the {role} performs the action",
        "Then the system fulfills it within the NFR latency budget",
    ]
    return UserStory(
        story_id=f"ST-{abs(hash(want)) % 10000:04d}",
        role=role, want=want, benefit=benefit, acceptance=acceptance,
    )
```

(3) Footer code

```python
# api.py — FastAPI route
from fastapi import FastAPI
from spec_generator import generate_story
app = FastAPI()

@app.post("/spec")
def create_spec(intake: dict):
    story = generate_story(intake)
    return story.__dict__
```

(4) Use Case Example — A queued intake item is posted to `/spec`; the generator returns a structured story with role, want, benefit, and Gherkin acceptance criteria, ready for backlog insertion.

(5) Step-by-Step Example 1. Architect triages intake from the queue (FC-MW-01). 2. POST the triaged object to `/spec`. 3. `generate_story()` builds INVEST story + Gherkin AC. 4. Story persisted; RTM row added linking story → stakeholder → test. 5. Story pushed to Jira (FN-06) — now developer-ready.

2.5 Technical Requirements

2.5.1 BUSINESS CASES

BC-02: *"Each developer-ready story eliminates an average 1.7 clarification cycles, saving ~40 minutes of cross-team coordination per story."*

2.5.2 EDGE CASES

- Intake in a non-English locale → generator routes to localization NFR path.
- Story fails INVEST "Small" check (estimate > 8 points) → auto-flagged for splitting.
- Duplicate intake (same hash) → merged with a traceability note.

2.5.3 USE CASES

UC-02 Generate Development Spec: triaged intake → INVEST story + Gherkin AC → RTM update → Jira push.

2.5.4 'TECHNICAL REQUIREMENTS' CARD

TECH REQ CARD · TR-VOC-01

Title: INVEST compliance gate **Type:** Functional + Process Quality **Statement:** *Every generated story shall satisfy all six INVEST criteria before backlog insertion; stories estimated above 8 points shall be flagged for splitting.* **Acceptance (Gherkin):**

```
Feature: INVEST gate
  Scenario: Oversized story is flagged
    Given a generated story estimated at 13 points
    When it is submitted to the backlog
    Then the system flags it "Too large — split before commit"
```

Priority: MUST · **Verification:** Unit test + PO review

2.5.5 TECH REQUIREMENTS INDEX + PRIORITY

ID	Requirement	Priority	Method
TR-VOC-01	INVEST compliance gate	MUST	XP test-first
TR-VOC-02	Gherkin AC auto-generation	MUST	Scrum/BDD
TR-VOC-03	RTM persistence per story	SHOULD	BABOK trace
TR-VOC-04	Localization routing	COULD	Lean option
TR-VOC-05	AI acceptance-criteria suggestion	WON'T (R1)	Lean experiment

3 Product Owner (Product Spec Document)

Per the **2020 Scrum Guide**, the Product Owner is accountable for **maximizing the value** of the product and for **Product Backlog management**: developing and communicating the Product Goal, creating and ordering backlog items, and ensuring the backlog is transparent

and understood. The Product Spec turns the VOC backlog into a value-ordered, release-mapped plan. HighTower.Digital fills this role as a **Team Leader @ Dev, Exec, Client** under the Partner/Emergency tiers.

3.1 Validated Design Document

3.1.1 STORY INDEX

The PO maintains a single, ordered Product Backlog. Index excerpt for the reference product:

Rank	Story	Product Goal contribution	WSJF
1	ST-03 Route intake to architect	Enables the SLA promise	9.4
2	ST-04 Generate spec from intake	Core value engine	8.8
3	ST-01 Submit a requirement	Entry point	7.1
4	ST-02 Live SLA countdown	Trust/visibility	4.2
5	ST-05 AI criteria suggestion	Delighter	2.0

3.1.2 STORY BACKLOG

Ordered by WSJF (Appendix C). The PO continuously refines the backlog so the top items are **"Ready"** (small, clear, testable) for the next Sprint Planning.

3.1.3 STORY LIST (FEATURES / FUNCTIONS)

The PO owns the **Definition of Ready** and **Definition of Done**, ensuring each story carries acceptance criteria, NFR links, and a value hypothesis before commitment.

3.2 Frontend Feature List

The PO prioritizes a **Release Roadmap View** so executives can see value delivery against the Product Goal.

3.2.1 FEATURE CARD — ROADMAP / VALUE BOARD

FRONTEND · FC-FE-PO-01

Feature: Release Roadmap & Value Board (React/TS)

(1) Header code

```
import { useRoadmap } from "../hooks/useRoadmap";
interface ReleaseColumn { id: string; goal: string; stories: string[]; }
```

(2) Function code

```
export function ValueBoard() {
  const { columns } = useRoadmap();            // ordered by WSJF
  return (
    <div className="board">
      {columns.map((c: ReleaseColumn) => (
        <section key={c.id} aria-label={c.goal}>
          <h4>{c.goal}</h4>
          <ol>{c.stories.map(s => <li key={s}>{s}</li>)}</ol>
        </section>
      ))}
    </div>
  );
}
```

(3) Footer code

```
export { ValueBoard } from "./ValueBoard";
```

(4) Use Case Example — An executive opens the Value Board to confirm the next release advances the stated Product Goal.

(5) Step-by-Step Example 1. PO orders backlog by WSJF. 2. Roadmap groups stories into release columns by Product Goal. 3. Exec reviews; PO captures feedback as new backlog items.

3.3 Middleware Features List

3.3.1 FEATURE CARD — BACKLOG ORDERING SERVICE

MIDDLEWARE · FC-MW-PO-01

Feature: Backlog Ordering Service (Node.js) — computes WSJF and returns the ordered backlog.

(1) Header code

```
interface BacklogItem { id: string; costOfDelay: number; jobSize: number; }
```

(2) Function code

```
export function orderByWsjf(items: BacklogItem[]): BacklogItem[] {
  return [...items].sort(
    (a, b) => (b.costOfDelay / b.jobSize) - (a.costOfDelay / a.jobSize)
  );
}
```

(3) Footer code

```
export { orderByWsjf };
```

(4) Use Case Example — The Value Board requests the ordered backlog; the service returns items sorted by WSJF.

(5) Step-by-Step Example — Fetch items → compute CoD/JobSize → sort descending → return to frontend.

3.4 Backend Feature List

3.4.1 FEATURE CARD — VALUE HYPOTHESIS TRACKER

BACKEND · FC-BE-PO-01

Feature: Value Hypothesis Tracker (Python) — records the measurable hypothesis behind each story and its validated outcome (Lean Build–Measure–Learn).

(1) Header code

```python
from dataclasses import dataclass
@dataclass
class Hypothesis:
    story_id: str
    metric: str
    target: float
    actual: float | None = None
```

(2) Function code

```python
def is_validated(h: Hypothesis) -> bool:
    return h.actual is not None and h.actual >= h.target
```

(3) Footer code

```python
__all__ = ["Hypothesis", "is_validated"]
```

(4) Use Case Example — After a release, the PO records the actual metric; validated hypotheses justify continued investment.

(5) Step-by-Step Example — Define hypothesis at commit → ship → measure → mark validated/ invalidated → feed next backlog ordering.

3.5 Technical Requirements

3.5.1 BUSINESS CASES

BC-03: *"WSJF ordering increases delivered value per sprint by prioritizing high Cost-of-Delay, low-effort items first."*

3.5.2 EDGE CASES

- Two items with identical WSJF → tie-break by stakeholder priority then FIFO.
- Story with no value hypothesis → blocked from "Ready."
- Job size = 0 → rejected (prevents divide-by-zero in WSJF).

3.5.3 USE CASES

UC-03 Order & Communicate Backlog: PO refines → orders by WSJF → publishes roadmap → captures exec feedback.

3.5.4 'TECHNICAL REQUIREMENTS' CARD

TECH REQ CARD · TR-PO-01

Title: Definition of Ready enforcement **Statement:** *No story shall be eligible for Sprint Planning unless it has acceptance criteria, a value hypothesis, and an estimate ≤ 8 points.* **Priority:** MUST · **Verification:** Backlog policy check + PO sign-off

3.5.5 TECH REQUIREMENTS INDEX + PRIORITY

ID	Requirement	Priority	Method
TR-PO-01	Definition of Ready enforcement	MUST	Scrum
TR-PO-02	WSJF backlog ordering	MUST	SAFe/Lean
TR-PO-03	Value hypothesis per story	SHOULD	Lean
TR-PO-04	Release roadmap view	SHOULD	Scrum
TR-PO-05	Forecast/burnup analytics	COULD	Scrum metrics

4 Technical Owner (Technical Spec Document)

The Technical Owner is the **Technical Leader @ Enterprise** accountable for the architecture, the **non-functional requirements (NFRs)**, and the **Architecture Decision Records (ADRs)**. The Technical Spec is the ISO/IEC/IEEE 29148 **SRS (Software Requirements Specification)** layer: it specifies API schemas, data models, NFR budgets, and edge-case behavior so that, per HighTower.Digital's promise, there is **"zero room for developer interpretation."**

4.1 Validated Design Document

4.1.1 STORY INDEX

Technical enablers and their NFR linkage:

Story	Technical concern	NFR target
TE-01	Idempotent intake API	Reliability 99.9%
TE-02	Spec generation latency	Performance ≤ 2 s p95
TE-03	Real-time SLA stream	Latency ≤ 10 s refresh
TE-04	Horizontal scale	1M records/min ingest

4.1.2 STORY BACKLOG

Technical stories are interleaved with feature stories so architecture keeps pace (XP's *continuous design* / *simple design*).

4.1.3 STORY LIST (FEATURES / FUNCTIONS)

Each technical story carries an **ADR** capturing context, decision, and consequences.

4.2 Frontend Feature List

4.2.1 FEATURE CARD — REAL-TIME SLA STREAM (CLIENT)

FRONTEND · FC-FE-TO-01

Feature: Real-time SLA countdown via WebSocket (React/TS) — meets the ≤10 s refresh NFR for real-time data systems.

(1) Header code

```
import { useEffect, useState } from "react";
```

(2) Function code

```
export function SlaCountdown({ ticketId }: { ticketId: string }) {
  const [remaining, setRemaining] = useState<number>(0);
  useEffect(() => {
    const ws = new WebSocket(`wss://api.hightower.digital/sla/${ticketId}`);
    ws.onmessage = e => setRemaining(JSON.parse(e.data).secondsLeft);
    return () => ws.close();
  }, [ticketId]);
  return <span role="timer">{Math.max(0, remaining)}s to SLA breach</span>;
}
```

(3) Footer code

```
export { SlaCountdown };
```

(4) Use Case Example — A client watches the live countdown; the architect sees the same stream and prioritizes accordingly.

(5) Step-by-Step Example — Open ticket → subscribe to `wss://.../sla/{id}` → render seconds-left → auto-close on unmount.

4.3 Middleware Features List

4.3.1 FEATURE CARD — IDEMPOTENT INTAKE GATEWAY

MIDDLEWARE · FC-MW-TO-01

Feature: Idempotent intake gateway (Node.js) — uses an `Idempotency-Key` header so retries never create duplicates (reliability NFR).

(1) Header code

```
import { Request, Response, NextFunction } from "express";
const seen = new Map<string, string>(); // replace with Redis in prod
```

(2) Function code

```
export function idempotency(req: Request, res: Response, next: NextFunction) {
  const key = req.header("Idempotency-Key");
  if (!key) return res.status(400).json({ error: "Idempotency-Key required" });
  if (seen.has(key)) return res.status(200).json({ ticketId: seen.get(key) });
  res.locals.idempotencyKey = key;
  next();
}
```

(3) Footer code

```
export { idempotency };
```

(4) Use Case Example — A flaky network causes the client to retry; the gateway returns the original ticket ID instead of creating a second one.

(5) Step-by-Step Example — Read key → if seen, return cached result → else proceed and cache on success.

4.4 Backend Feature List

4.4.1 FEATURE CARD — STREAMING INGEST PIPELINE

BACKEND · FC-BE-TO-01

Feature: Streaming ingest pipeline (Python) — async batch ingest meeting the 1M records/min scalability NFR.

(1) Header code

```python
import asyncio
from typing import AsyncIterator
```

(2) Function code

```python
async def ingest(stream: AsyncIterator[dict], batch_size: int = 5000) -> int:
    batch, total = [], 0
    async for record in stream:
        batch.append(record)
        if len(batch) >= batch_size:
            await persist_batch(batch)   # bulk insert
            total += len(batch); batch.clear()
    if batch:
        await persist_batch(batch); total += len(batch)
    return total
```

(3) Footer code

```python
__all__ = ["ingest"]
```

(4) Use Case Example — A real-time data source streams events; the pipeline batches and bulk-persists to sustain throughput without data loss.

(5) Step-by-Step Example — Open stream → accumulate batch → bulk persist at threshold → flush remainder → return total.

4.5 Technical Requirements

4.5.1 BUSINESS CASES

BC-04: *"Idempotency and bulk ingest prevent duplicate-record incidents that previously cost ~6 engineer-hours/month in reconciliation."*

4.5.2 EDGE CASES

- Missing `Idempotency-Key` → 400 (forces correct client behavior).
- WebSocket drops mid-stream → client auto-reconnects with backoff.
- Batch persist partial failure → transactional rollback; record-level dead-letter queue.

4.5.3 USE CASES

UC-04 Guarantee Reliable Throughput: ingest stream → batch → persist transactionally → expose live SLA via WebSocket.

4.5.4 'TECHNICAL REQUIREMENTS' CARD

TECH REQ CARD · TR-TO-01

Title: NFR budget — spec generation latency **Statement:** *95th-percentile spec generation latency shall be ≤ 2 seconds at 1,000 concurrent requests; real-time SLA stream shall refresh ≤ 10 seconds.*
Priority: MUST · **Verification:** Load test (k6) + p95 dashboard

4.5.5 TECH REQUIREMENTS INDEX + PRIORITY

ID	Requirement	Priority	Method
TR-TO-01	Spec latency ≤2 s p95	MUST	XP CI perf gate
TR-TO-02	Idempotent intake API	MUST	Scrum/XP
TR-TO-03	1M records/min ingest	SHOULD	Lean flow
TR-TO-04	Real-time SLA stream ≤10 s	SHOULD	Scrum
TR-TO-05	ADR for every architecture decision	MUST	ADR standard

5 Domain Owner (Domain / Brand Spec Document)

The Domain Owner sits in **Brand Management** @ **Enterprise** and is accountable for **domain integrity and brand consistency** across every requirement. Grounded in **Domain-Driven Design (DDD)** — bounded contexts, ubiquitous language, and a shared domain model — the Domain/Brand Spec ensures terminology, tone, accessibility, and visual identity are encoded as requirements, not afterthoughts. For HighTower.Digital this protects the "Create Excellence!!" brand promise and the shield identity across all client-facing surfaces.

5.1 Validated Design Document

5.1.1 STORY INDEX

Story	Domain concern	Standard
DM-01	Ubiquitous language enforcement	DDD
DM-02	Brand token system (color/type)	Design system
DM-03	Accessibility (WCAG 2.2 AA)	WCAG
DM-04	Bounded-context API naming	DDD

5.1.2 STORY BACKLOG

Domain stories are cross-cutting; the Domain Owner reviews every other backlog for terminology and brand conformance before "Done."

5.1.3 STORY LIST (FEATURES / FUNCTIONS)

The **ubiquitous language glossary** (Appendix D) is the single source of truth; code, UI, and docs must use the same terms.

5.2 Frontend Feature List

5.2.1 FEATURE CARD — BRAND TOKEN PROVIDER

FRONTEND · FC-FE-DM-01

Feature: Brand token provider (React/TS) — injects HighTower.Digital design tokens (navy/blue/cyan/purple) so every surface is on-brand by default.

(1) Header code

```
export const hdTokens = {
  navy: "#0a1f44", blue: "#1b4fd8", cyan: "#22b8e6", purple: "#7b4fe0",
} as const;
```

(2) Function code

```
import { createContext, useContext } from "react";
const BrandCtx = createContext(hdTokens);
export const useBrand = () => useContext(BrandCtx);
export const BrandProvider = ({ children }: { children: React.ReactNode }) =>
  <BrandCtx.Provider value={hdTokens}>{children}</BrandCtx.Provider>;
```

(3) Footer code

```
export { hdTokens, useBrand, BrandProvider };
```

(4) Use Case Example — Any component calls `useBrand()` to render with approved brand colors — no hardcoded hex values pass review.

(5) Step-by-Step Example — Wrap app in `BrandProvider` → components consume `useBrand()` → lint rule blocks raw hex outside tokens.

5.3 Middleware Features List

5.3.1 FEATURE CARD — UBIQUITOUS LANGUAGE VALIDATOR

MIDDLEWARE · FC-MW-DM-01

Feature: Ubiquitous-language validator (Node.js) — rejects API payloads using non-canonical domain terms (e.g., "ticket" vs "trouble-ticket").

(1) Header code

```
const canonical: Record<string,string> = { ticket: "troubleTicket", req: "requirement" };
```

(2) Function code

```
export function normalizeTerms(payload: Record<string, unknown>) {
  const out: Record<string, unknown> = {};
  for (const [k, v] of Object.entries(payload)) out[canonical[k] ?? k] = v;
  return out;
}
```

(3) Footer code

```
export { normalizeTerms };
```

(4) Use Case Example — A legacy client sends `{ ticket: ... }`; middleware normalizes to the ubiquitous `troubleTicket` before persistence.

(5) Step-by-Step Example — Receive payload → map keys to canonical terms → forward normalized object downstream.

5.4 Backend Feature List

5.4.1 FEATURE CARD — BOUNDED-CONTEXT REGISTRY

BACKEND · FC-BE-DM-01

Feature: Bounded-context registry (Python) — declares which domain owns which entity, preventing model leakage across contexts.

(1) Header code

```python
from dataclasses import dataclass, field
@dataclass
class BoundedContext:
    name: str
    entities: set[str] = field(default_factory=set)
```

(2) Function code

```python
def owns(ctx: BoundedContext, entity: str) -> bool:
    return entity in ctx.entities
```

(3) Footer code

```python
__all__ = ["BoundedContext", "owns"]
```

(4) Use Case Example — A service checks `owns(billing_ctx, "Invoice")` before mutating an entity, enforcing context boundaries.

(5) Step-by-Step Example — Register contexts → assign entities → guard mutations with `owns()` checks.

5.5 Technical Requirements

5.5.1 BUSINESS CASES

BC-05: *"Consistent ubiquitous language and brand tokens reduce onboarding time and prevent costly brand/compliance rework across client deliverables."*

5.5.2 EDGE CASES

- Unmapped term submitted → flagged for glossary review, not silently passed.
- Color contrast below WCAG 2.2 AA → blocked at design-token lint.
- Entity claimed by two contexts → registry raises a boundary conflict.

5.5.3 USE CASES

UC-05 Enforce Domain & Brand Integrity: validate terms → enforce brand tokens → check accessibility → guard context boundaries.

5.5.4 'TECHNICAL REQUIREMENTS' CARD

TECH REQ CARD · TR-DM-01

Title: Brand & accessibility conformance gate **Statement:** *All client-facing surfaces shall use approved brand tokens and meet WCAG 2.2 AA contrast; raw hex values and sub-AA contrast shall fail CI.*
Priority: MUST · **Verification:** Design-token lint + automated a11y test (axe)

5.5.5 TECH REQUIREMENTS INDEX + PRIORITY

ID	Requirement	Priority	Method
TR-DM-01	Brand & a11y conformance gate	MUST	XP CI
TR-DM-02	Ubiquitous language validator	MUST	DDD
TR-DM-03	Bounded-context registry	SHOULD	DDD
TR-DM-04	Design token system	SHOULD	Design system
TR-DM-05	Brand telemetry/audit	COULD	Lean metrics

Appendix A — Methods: Agile Scrum + Lean + XP

HighTower.Digital runs IT Requirements As-A-Service on a hybrid of three complementary agile schools. **Scrum** provides the cadence and roles; **Lean** provides flow, waste elimination, and Build–Measure–Learn validation; **XP** provides the engineering discipline (test-first, continuous integration, simple design) that keeps specs developer-ready.

Dimension	Scrum	Lean	Extreme Programming (XP)
Core unit	Time-boxed Sprint	Continuous flow	Short iterations
Key roles	PO, Scrum Master, Developers	Flow/value stream owner	Whole team, on-site customer
Requirements form	Product Backlog (stories)	Pull-based options	Story cards + tests
Prioritization	PO value ordering	Cost of Delay / WSJF	Customer-chosen stories
Quality practice	Definition of Done	Build quality in	TDD, pair programming, CI
Feedback loop	Sprint Review	Build–Measure–Learn	Continuous (CI + customer)
HDI use	Cadence + ownership	Triage flow + validation	Spec verifiability + automation

Why hybrid? Scrum alone orders work but does not guarantee engineering quality; Lean optimizes flow but is method-agnostic about specs; XP enforces verifiable, test-first requirements. Together they realize the HighTower.Digital promise of **sprint-ready, zero-translation-loss specifications**.

Appendix B — Requirements Standards & Quality Criteria

ISO/IEC/IEEE 29148:2018 (which obsoleted IEEE 830-1998) defines the requirements engineering life-cycle and the document set used throughout this report: **BRS** (Business), **StRS** (Stakeholder — used in §1), **SyRS** (System), **SRS** (Software — used in §4), and **OpsCon**. The standard's characteristics of a good requirement anchor every Technical Requirements Card:

- **Necessary** — defines an essential capability; removing it leaves a deficiency.
- **Unambiguous** — one interpretation only.
- **Complete** — needs no further amplification.
- **Singular** — one requirement, no conjunctions.
- **Feasible** — achievable within constraints.
- **Verifiable** — can be proven by test, demo, inspection, or analysis.
- **Correct** — accurately represents the real need.
- **Conforming** — follows the approved template/standard.

Complementary bodies of knowledge: **IIBA BABOK® Guide v3** (elicitation, requirements life-cycle management, traceability), the **2020 Scrum Guide** (Product Owner accountability, Product Backlog), **Kano model** and **QFD** (VOC weighting), **INVEST** (story quality), **Gherkin/BDD** (executable acceptance criteria), **Domain-Driven Design** (bounded contexts, ubiquitous language), **Architecture Decision Records (ADR)**, and **WCAG 2.2 AA** (accessibility).

Appendix C — Prioritization Frameworks

Framework	Formula / Buckets	Best for	Used in
MoSCoW	Must / Should / Could / Won't	Scope negotiation, releases	All Tech Req indexes
WSJF	Cost of Delay ÷ Job Size	Sequencing the backlog	§2.1.2, §3
RICE	(Reach × Impact × Confidence) ÷ Effort	Comparing initiatives	Portfolio planning

Worked WSJF (ST-03): Cost of Delay components — user/business value (high), time criticality (high, SLA-bound), risk reduction (high) ≈ 47; Job Size ≈ 5 ⇒ **WSJF ≈ 9.4**, placing it at rank 1.

Appendix D — Glossary (Ubiquitous Language)

Term	Definition
Trouble-Ticket	Canonical term for a submitted requirement/incident entering the SLA pipeline.
Developer-Ready Spec	A specification with API schema, data model, acceptance criteria, and edge cases — zero open interpretation.
SLA Tier	Advisory (4 h), Partner (1 h), Emergency Owner (15 min) response thresholds.
RTM	Requirements Traceability Matrix linking need → story → test.
NFR	Non-Functional Requirement (performance, scalability, reliability, security, etc.).
ADR	Architecture Decision Record — context, decision, consequences.
Bounded Context	DDD boundary within which a domain model and its terms are consistent.
INVEST	Independent, Negotiable, Valuable, Estimable, Small, Testable.
WSJF	Weighted Shortest Job First — Cost of Delay ÷ Job Size.

Appendix E — References

1. ISO/IEC/IEEE 29148:2018 — *Systems and software engineering — Life cycle processes — Requirements engineering.* IEEE SA. https://standards.ieee.org/standard/29148-2018.html
2. ReqView — *ISO/IEC/IEEE 29148 Requirements Specification Templates (StRS/SyRS/SRS/BRS/OpsCon).* https://www.reqview.com/doc/iso-iec-ieee-29148-templates
3. Schwaber, K. & Sutherland, J. — *The 2020 Scrum Guide.* https://scrumguides.org/scrum-guide.html
4. IIBA — *A Guide to the Business Analysis Body of Knowledge (BABOK® Guide v3).* https://www.iiba.org/
5. Blank, S. — *Jobs-To-Be-Done as the Front End of Customer Discovery.* https://steveblank.com/2021/11/04/market-definition-its-the-front-end-of-customer-discovery/
6. Ulwick, T. — *Jobs to Be Done (JTBD): The Original Framework.* Strategyn. https://strategyn.com/jobs-to-be-done/
7. Fitzpatrick, R. — *The Mom Test* (customer discovery interview method).
8. AltexSoft — *Nonfunctional Requirements: Examples, Types and Best Practices.* https://www.altexsoft.com/blog/non-functional-requirements/

9. TestQuality — *Gherkin User Stories & Acceptance Criteria Guide (2026).* https://testquality.com/gherkin-user-stories-acceptance-criteria-guide
10. Boost / PlatinumEdge — *INVEST Criteria for Agile User Stories.* https://www.boost.co.nz/blog/2021/10/invest-criteria
11. Architecture Decision Records. https://adr.github.io/
12. KanbanZone — *Agile Frameworks Compared: Scrum, Kanban, Lean, XP (2024).* https://kanbanzone.com/2024/agile-frameworks-compared/
13. PPM Express — *From RICE to WSJF: Prioritization Techniques.* https://www.ppm.express/blog/13-prioritization-techniques
14. GetProductPeople — *Prioritization Techniques: RICE, MoSCoW, ICE & Kano.* https://www.getproductpeople.com/blog/prioritization-techniques-rice-moscow-ice-kano
15. HighTower.Digital — *MSP SLA #1+5 — IT Requirements As-A-Service.* https://www.hightower.digital/msp-sla-15

Document control. HDI Confidential · Author: LW Atkinson (CEO/CTO) · Version R1.A · 06/25/2026.