WHITEPAPER

# Prototype to Experiment, Framework to Solidify

*A Visionary Software Solutions philosophy for choosing prototypes, mapping adjacency, and converging on durable frameworks*

Visionary Software Solutions

Internal methodology — March 2026

**Abstract**

Small software firms routinely lose calendar time to the wrong kind of exploration: polishing throwaway prototypes when the solution is already clear, or prematurely frameworking when the product shape, interaction model, and problem boundaries are still opaque. This document states Visionary Software Solutions' (VSS) core design philosophy—*prototype when you must experiment; build a framework when you can already see the solution in its fullness*—and lays out a repeatable procedure for moving from a crisp core idea through adjacency mapping, quantitative opportunity scoring, technology exploration, convergence on a shared core stack, and an evolving framework description that matures into a formal Framework Design Document (FDD). A running example—a voice-first to-do tool—anchors each stage. A section on evaluation of risk explains why this philosophy centers constructive design and placement notes over formal risk matrices at this scale, while distinguishing where full risk frameworks remain essential. The closing section addresses the high-leverage case where disciplined mental modeling can substitute for an early prototype, allowing a team to jump directly to framework construction without wasteful iteration loops.

**Keywords:** software methodology; prototyping; framework design; adjacency mapping; product psychology; product risk; small software firms; Visionary Software Solutions

## Contents

1 Manifesto and dual formulation ..... 2
2 Prototyping: when uncertainty dominates ..... 2
2.1 The problem prototypes solve ..... 2
2.2 What a prototype is, in this philosophy ..... 2
2.3 Relationship to later framework work ..... 2
3 Framework building: an end-to-end procedure ..... 3
3.1 Stage A — Anchor the core problem and defining feature ..... 3
3.2 Stage B — First ring: technically and purposefully adjacent facets ..... 3
3.3 Stage C — Expand arenas: platforms, industries, and "unrelated" adjacency ..... 3
3.4 Stage D — Qualitative map and scientific scoring ..... 4
4 The adjacency and opportunity matrix ..... 4
4.1 Mindmap (qualitative synthesis) ..... 4
4.2 Tabular extraction ..... 4
4.3 Weight definitions ..... 5
4.4 Worked scores (illustrative) ..... 5
4.5 Drawing conclusions ..... 6

5 Technology exploration ..... 6
6 Core technology convergence ..... 6
7 Framework description and path to an FDD ..... 7
8 When mental design replaces an early prototype ..... 7
9 Evaluation of risk in this philosophy ..... 7
10 Conclusion ..... 8

# 1 Manifesto and dual formulation

The same principle admits a short and a longer phrasing:

- **Short form:** "Prototype to experiment, framework to solidify."
- **Expanded form:** "Prototype if you feel the need to experiment; build a framework if you already envision the solution in its fullness."

Both formulations are intentional. The first is memorable on a wall or slide; the second names the decision criterion—**uncertainty about the shape of the solution** versus **clarity sufficient to invest in durable structure**.

# 2 Prototyping: when uncertainty dominates

## 2.1 The problem prototypes solve

When you do not yet know how a solution will **look and feel**, which complex interactive behaviors it requires, or which architectural approach best decomposes a hard problem, a **prototype** is the appropriate tool. Prototypes are instruments of **epistemic reduction**: they turn unknowns into observations you can react to.

## 2.2 What a prototype is, in this philosophy

In VSS usage, a **prototype** is a dedicated, one-off implementation built for **real use** in a narrow context. It is **not** designed upfront for general reuse, wholesale adaptation, or multi-tenant extension. It exists to answer "What is it like to use this?" and "Does this decomposition work?" for **the problem in front of you right now**.

You may iterate the prototype aggressively and let it grow messier as it absorbs learning. That mess is acceptable—indeed expected—because the artifact's job is to buy insight, not to model your long-term engineering standards.

## 2.3 Relationship to later framework work

Over time, a successful prototype may inform—or partially feed—a framework. The path is often **prototype → lessons → clean-room framework**, not **prototype gradually becomes the framework without a break**. The break matters: frameworks need boundaries, invariants, and documented extension points; prototypes optimized for speed rarely arrive with those intact.

# 3 Framework building: an end-to-end procedure

After prototyping (or after equivalent clarity obtained another way—see Section 8), framework building proceeds in **stages**. The subsections below describe each stage in detail and, within each stage, walk the **voice-first to-do** example summarized here:

> **Running example — voice-first to-do**
>
> Capture tasks quickly by voice, with minimal friction, for an individual user who needs speed over structure.

## 3.1 Stage A — Anchor the core problem and defining feature

**Goal.** State the smallest honest description of what is uniquely valuable.

> **Voice-first to-do — Stage A**
>
> Core problem: **capturing actionable items faster than typing allows in motion or split-attention contexts.** Defining feature: **reliable, low-latency voice capture that becomes structured list items** —not generic task management, not enterprise workflow—unless you later choose to expand there deliberately.

**Practice.** Write one paragraph that could not apply verbatim to a competitor's generic to-do app without lying. That paragraph is your center node for everything that follows.

## 3.2 Stage B — First ring: technically and purposefully adjacent facets

**Goal.** Enumerate capabilities that sit immediately around the core in the same product surface: same user, same session, nearby jobs-to-be-done.

> **Example ring — product adjacencies**
>
> Voice disambiguation ("buy milk" vs. "remind me about buying milk"), list organization (projects, tags), reminders and notifications, capture while offline, quick corrections, attachments or links, shared lists for a household team.

**Mapping note.** On a mindmap, these are nodes one hop from the center, edges labeled by relationship type: **data dependency**, **UX dependency**, **same workflow step**, etc.

## 3.3 Stage C — Expand arenas: platforms, industries, and "unrelated" adjacency

**Goal.** Move outward: adjacent technologies (speech SDKs, on-device ML, push infra), adjacent platforms (watch, car, desktop assistant), adjacent industries (field service, clinical rounding, education), and **seemingly unrelated** industries where the **same underlying capability** might apply if repositioned.

> **Example expansions — arenas and industries**
>
> Construction job-site coordination (many crews, noisy environment, hands busy), healthcare rounding (hands full, compliance context), retail back-of-house (shift tasks), logistics dispatch. The **unrelated** scan is deliberate: voice-first capture of structured units of work is not industry-specific; only positioning and compliance are.

### 3.4 Stage D — Qualitative map and scientific scoring

This stage is where intuition becomes reviewable. Section 4 formalizes scoring; here, the activity is to **produce artifacts**: a mindmap for synthesis, a table for comparison, and scores for prioritization.

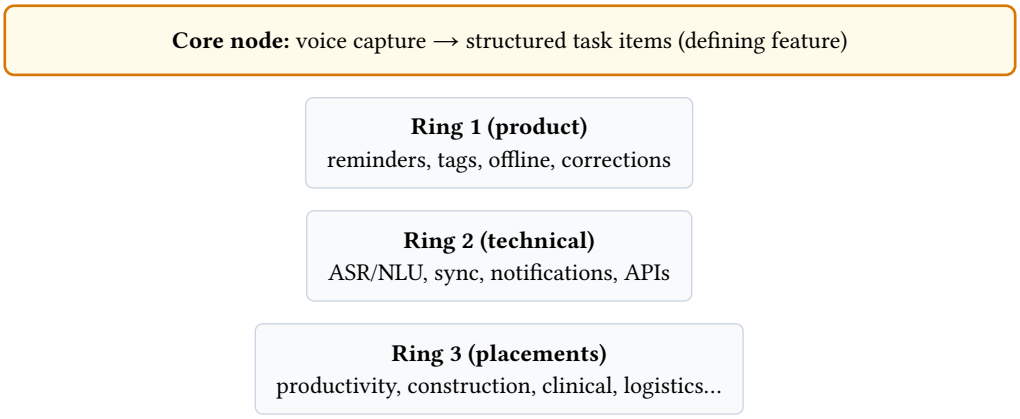


Figure 1: Conceptual mindmap layout for the voice-first to-do example (center ⟶ rings ⟶ industries).

> **Voice-first to-do — qualitative walkthrough**
>
> Start with a mindmap whose center is "voice ⟶ structured tasks." Branch to technical pillars (ASR, NLU, sync, UI), user modes (single user, small team), and industry spokes (personal productivity, construction PM, nursing). For each leaf you might plausibly serve, you will later assign weights (adjacency, complexity delta, user importance, business composite).

## 4 The adjacency and opportunity matrix

This section specifies VSS's **scientific framework** for turning qualitative exploration into comparable, revisable decisions. It is designed for small teams without a dedicated strategy office: the math is lightweight, the discipline is in keeping the sheet current.

### 4.1 Mindmap (qualitative synthesis)

Use any tool you prefer. The map is not the source of truth—it is a **thinking surface**. Conventions that work well:

- **Center:** core problem + defining feature (one node).
- **First ring:** same-product adjacencies.
- **Second ring:** platform and technical dependencies.
- **Third ring:** industry and persona hypotheses.
- **Dashed edges:** speculative or low-confidence links.

Export or snapshot the map for workshops; version it like code when decisions shift.

### 4.2 Tabular extraction

Periodically **flatten** the map into a table—manual entry is fine. Each row is a **placement hypothesis**: a market or use case where your core technology might land. Minimum columns:

| Placement | Persona / context | Core fit narrative | Notes |
|---|---|---|---|
| Personal productivity | Individual, mobile | Voice beats typing for capture in motion | Baseline |
| Construction PM | Site / multi-crew | Hands-busy; many parallel tasks | Loud environment ASR risk |
| Clinical rounding | Nurse / physician | Interrupt-driven capture | Compliance, PHI |

Table 1: Minimum columns for adjacency extraction (illustrative).

## 4.3 Weight definitions

For each placement row, maintain quantitative fields (normalized to, say, 0–10):

- **Adjacency** $A$: conceptual closeness to the original defining feature and core user job.
- **Complexity delta** $C$: incremental engineering and operational complexity versus the core feature set (higher means harder).
- **User importance** $U$: severity of the problem **within that industry** for the target persona (evidence from interviews, proxies, or informed estimates—label uncertainty explicitly).
- **Business composite** $B$: a composite of projected **revenue potential**, **profit margin**, **scalability**, and **ease of implementation** via translation or extension of your core technology.

Let revenue, margin, scalability, and implementation ease be $R, M, S, E$ on the same scale. A transparent default is an equally weighted mean:

$$B = \frac{R + M + S + E}{4}$$

Teams may use weighted averages if their strategy prioritizes margin over revenue in early years, etc.; the critical rule is **one agreed formula per program**, documented beside the sheet.

Optionally define a **priority index** that penalizes complexity and rewards adjacency and business fit, e.g.:

$$P = \alpha A + \beta B + \gamma U - \delta C$$

with nonnegative coefficients summing sensibly and tuned once—not per row—to avoid overfitting optimism.

## 4.4 Worked scores (illustrative)

The numbers below are **pedagogical**, not market research outputs:

| Placement | $A$ | $C$ | $U$ | **Components** $(R, M, S, E)$ | $B$ |
|---|---|---|---|---|---|
| Personal productivity | 10 | 3 | 9 | (7,8,9,9) | 8.25 |
| Construction PM | 8 | 6 | 8 | (8,10,10,9.8) | 9.45 |
| Clinical rounding | 6 | 9 | 10 | (9,7,8,5) | 7.25 |

Table 2: Illustrative scoring for three placements (voice-first to-do core).

### 4.5 Drawing conclusions

Conclusions should read like **actionable theses**, not vague positivity. A good conclusion ties **placement**, **persona pain**, **fit narrative**, and **composite score** to a **go / watch / defer** stance.

> **Sample conclusion — construction placement**
>
> The construction industry needs fast task planning and tracking under field conditions. Project managers coordinate many crews and moving tasks; entering work into a robust system by voice can increase management speed and reduce cognitive load versus stopping to type. With a business composite of approximately 9.45 on a 0–10 scale (illustrative components: strong margin and scalability, strong implementation fit via the same voice-capture core), an adjacent product exploration targeting construction PM workflows is **recommended for structured discovery**—interviews, compliance checklist, and a focused technology spike on noisy-environment speech recognition before commitment.

## 5 Technology exploration

Once placements are prioritized, **technology exploration** asks: for each **adjacent use case of interest**, what stacks, modalities, constraints, and non-functional requirements apply?

**Activities.**

- List **modalities**: mobile, wearable, vehicle, desktop, shared kiosk, mixed.
- List **constraints**: offline, air-gapped, HIPAA, SOC2, regional data residency, intermittent connectivity on job sites.
- List **integration surfaces**: calendar, SMS, email ingestion, ERP APIs, identity providers.
- Capture **preferences** where they exist: e.g., on-site crews may prefer push-to-talk over always-listening for privacy and battery.

> **Voice-first to-do — technology exploration**
>
> Exploration might surface: on-device ASR for latency; server-side models for vocabulary customization; SMS/email as alternate capture channels for users who cannot speak; different list models for personal vs. crew-based tasks.

## 6 Core technology convergence

From the union of explorations, identify the **smallest set of technologies, patterns, procedures, and user workflows** that cover the largest **feasible** slice of prioritized placements without collapsing into an unbuildable monolith.

**Output.** A shortlist of **shared kernels**: e.g., a single "capture and normalize utterance ⟶ task object" pipeline; a single sync model; a policy layer for permissions that scales from individual to team without forked codebases.

> **Voice-first to-do — core convergence**
>
> Convergence might dictate: one task engine; multiple ingress adapters (voice, typed mobile, SMS, email); one reminder subsystem; industry-specific **policy packs** rather than industry-specific engines.

## 7 Framework description and path to an FDD

**Framework description** begins as an **unofficial** living outline: bullet lists, diagrams, interface sketches, and explicit non-goals. It matures into an **official Framework Design Document** when invariants, module boundaries, versioning, and extension mechanisms are stable enough to serve as a contract for implementation.

Minimum contents as it hardens:

- **Problem statement and defining feature** (carried from Stage A).
- **Module graph** and responsibilities.
- **Technology stack** choices with rationale and escape hatches.
- **Interfaces**: user-facing surfaces **and** APIs between modules; external integrations.
- **Scalability and operability**: data model evolution, deployment topology, observability.
- **Security and compliance hooks** even if the first vertical is low-regulation.

> **Voice-first to-do — framework sketch**
>
> Interfaces might explicitly include: voice capture UI; typed entry; SMS and email ingestion endpoints; push notification services; REST or GraphQL for third-party automation. The **core engine** specifies task lifecycle, deduplication, list membership, and reminder scheduling—technology-agnostic rules with concrete adapters underneath.

## 8 When mental design replaces an early prototype

The preceding sections describe **externalized** design work: maps, tables, spikes, and documents. There is a final, high-impact observation.

Sometimes, by applying sustained, structured thinking—walking the same stages **in thought**—you can traverse the full design space without building a disposable codebase first. If you can simulate the user journeys, foresee major technical risks, and anticipate where modules must flex for adjacency you care about, and your expected **development turbulence** is low, then **building the framework immediately** is rational.

Many teams default to a prototype because it feels safer than committing to structure. That habit becomes costly when the prototype only confirms what disciplined modeling would have predicted. In those cases, the prototype is not learning time; it is **avoidance of specification**.

**Conversely**, if you **cannot** see the interaction model, the failure modes, or the decomposition—if the problem is genuinely novel to you—then a prototype is not waste; it is the cheapest way to falsify bad assumptions.

The decision rule restated:

- **High uncertainty about product shape or feasibility** → prototype.
- **High clarity across UX, architecture, and adjacency strategy** → framework, with maps and scores as lightweight guardrails, not substitutes for thought.

## 9 Evaluation of risk in this philosophy

This document takes a **constructive** posture: it does not prescribe a parallel **risk-analysis framework** sitting alongside adjacency mapping and scoring. That omission is deliberate. The aim here is minimalism and conciseness of thought—enough structure to turn an idea into a defensible plan and to stretch that idea across related placements without drowning the team in ceremony.

**Where risk still belongs.** Risks should still be **noted and considered**. The natural place is in the notes you keep while working through **expanded product design**: each application area, each persona, each technical

or regulatory context. Treat those notes as first-class artifacts—short, dated, tied to a placement row or a module—not as a matrix that must be complete before anyone writes code. In that reflective space you will sometimes see **intractability** (a path that is not worth the cost) and sometimes **opportunity** (a cheap extension that unlocks a new wedge). Both are outcomes of honest attention; neither requires a formal risk register to be valuable.

**When full risk frameworks earn their place.** At higher levels of complexity and impact—safety-critical systems, large system-of-systems integrations, enterprise-grade products with contractual liability—structured risk methods, traceable hazards, and review gates are not optional. VSS uses and respects those practices where the domain demands them. The core philosophy in this paper is aimed at a different scale: small teams moving quickly to establish a sharp solution in market, with adjacency baked in early so many facets of a diverse customer base can be **accommodated in the design phase at the beginning** rather than bolted on after the fact.

**Why not center risk matrices here.** Formal risk matrices, in the fast-paced, fast-to-market arena this paper addresses, tend to **limit** thinking as often as they sharpen it. They invite defensive completeness when the priority is directional speed: refine the defining feature, map rings, score placements, and converge on a kernel. The caricature "if we build it, they will come" is rightly criticized; the stronger move is to build a **specific** solution to a **specific, sharp problem**, establish credibility and revenue, and use adjacency to prepare for what comes next—without letting premature negativity throttle exploration.

**Team psychology and language.** Positive language and forward framing foster a builder-focused culture: we are here to shape something that works. Negative language—warnings, doubts, catalogued failures—has a legitimate role in keeping expectations realistic and in passing conversation. When it becomes the default **frame** for how work is discussed, it introduces a back-pressure that slows thinking and visibly dampens progress across the team. Even a slight persistent tilt toward the skeptical register compounds. The balance this philosophy recommends: hold risks in notes and reviews where they inform decisions; do not let the risk conversation **become** the product culture for teams that need to ship.

**A simple decision hierarchy.** If the market need is real and you can build the solution, you should build it. Secondarily, if a **stronger or adjacent** market need is visible and your design can be adapted or expanded to capture it without losing the core, you should strongly consider doing so. Front-loaded effort on breadth of applicability—often on the order of ten to thirty percent beyond a single-vertical minimum—can position you to serve two additional same-sized placements alongside your first. In revenue and reach terms, that is roughly **three times** the capture of staying narrow, for a fraction of three times the total cost. The teams that invest that effort early widen the path for what comes later; that trade is often well worth making.

# 10 Conclusion

"Prototype to experiment, framework to solidify" is a compression of a richer process. Prototypes earn their keep when uncertainty is real; frameworks earn their keep when clarity allows shared investment in invariants and interfaces. Between those poles, VSS uses adjacency mapping, explicit scoring, technology exploration, convergence on a core kernel, and a maturing framework description—illustrated here with a voice-first to-do thread—to keep small-team decisions legible, comparable, and aligned with business reality.