Building for Contemplation: When Spiritual Practice Becomes Code

November 2025

10 min read • 2,313 words

"Study to shew thyself approved unto God, a workman that needeth not to be ashamed, rightly dividing the word of truth." — 2 Timothy 2:15 (KJV)

I've been building Bible study tools not because I'm particularly devout, but because I needed them. My spiritual practice—irregular, questioning, deeply personal—required tools that didn't exist. So I built kjvstudy.org, and in doing so, discovered something profound about the recursive loop between personal values and professional work.

Most Bible apps optimize for engagement. Daily streaks, social features, gamification mechanics—the same dark patterns that consume virtue across the internet. They treat Scripture like content to be consumed rather than wisdom to be contemplated. The algorithmic approach to spirituality feels like serving two masters, and we know how that story ends

> "No man can serve two masters: for either he will hate the one, and love the other; or else he will hold to the one, and despise the other." — Matthew 6:24 (KJV). The tension between engagement metrics and spiritual contemplation is irreconcilable.

.

What I needed was different: a quiet digital space for deep reading. No notifications. No social features. No gamification. Just the text, the scholarship, and the space to think. Building this revealed how profoundly our tools shape not just what we think about, but how we're capable of thinking at all.

The Personal Need That Became Code

My relationship with Scripture is complicated. Raised Christian, became agnostic, then found myself returning to biblical texts not for doctrine but for the recursive patterns—the same wisdom appearing in different forms, pointing toward something fundamental about consciousness and human experience.

The existing tools frustrated me. Commercial Bible apps interrupted reading with ads and push notifications. Academic resources hid behind paywalls and institutional barriers. Popular study sites optimized for quick answers rather than deep understanding. Everything treated Scripture as either product to monetize or content to consume.

I wanted what programmers call a "pure function"—input text, output understanding, no side effects:

```
def contemplate_scripture(passage, context=None):
    """
    A pure function for spiritual practice.
    No tracking, no metrics, no manipulation.
    Just text and consciousness meeting.
    """
    if context:
        passage = enrich_with_scholarship(passage, context)

    # No engagement tracking
    # No notification scheduling
    # No addiction mechanics

    return passage.presented_for_contemplation()
```

This wasn't about rejecting technology in spiritual practice. It was about building technology that serves contemplation rather than consuming attention. The difference matters more than most realize.

Values Manifesting as Architecture

Every technical decision in kjvstudy.org embodies a value about how technology should serve human consciousness. These aren't arbitrary implementation details—they're philosophical commitments expressed through code.

Choosing Depth Over Engagement

The platform provides seminary-level theological content—extensive Greek and Hebrew word studies, detailed historical context, comprehensive cross-references. Each of the nine Fruits of the Spirit has 4-5 paragraph expositions with etymological analysis. The seven theological study guides contain over 50 detailed sections total.

This depth serves a purpose: real understanding requires time and attention. You can't contemplating the nature of agape love in a TikTok-length snippet. Complex ideas need space to breathe

> Edward Tufte's design principles shaped the interface—generous whitespace, readable typography, sidenotes for context. The design itself becomes contemplative practice, creating cognitive space for deeper thinking.

.

```
class TheologicalContent:
    """Not snippets but scholarship."""

    def __init__(self, topic):
        self.greek_etymology = deep_linguistic_analysis(topic)
        self.hebrew_roots = trace_to_source(topic)
        self.historical_context = situate_in_time(topic)
        self.cross_references = find_recursive_patterns(topic)
        self.practical_application = bridge_to_present(topic)

        # Not sound bites but substance
        self.minimum_depth = "seminary_level"
        self.reading_time = "as_long_as_needed"
```

Genealogy as Pattern Recognition

One of the platform's most contemplative features is the biblical family tree—a searchable genealogy from Adam through the entire biblical narrative. This wasn't just a data visualization exercise. It became a meditation on recursive patterns across generations.

Following the lineage from Adam to Noah to Abraham to David to Solomon to Jesus reveals something profound: the same patterns repeat. Faithfulness and doubt. Obedience and rebellion. Divine promise and human failure. The genealogies aren't just historical records—they're consciousness patterns persisting across time.

Building the search functionality meant encoding these relationships in data structures. But the process revealed theological insights:

```
class BiblicalLineage:
    """Tracing consciousness patterns through generations."""

    def __init__(self, person):
        self.name = person
        self.parents = find_parents(person)
        self.children = find_children(person)
        self.notable_for = extract_significance(person)

    def trace_to_adam(self):
        """Every lineage traces back to origin."""
        # Recursive search through ancestors
        # Like consciousness tracing back to source
        current = self
        lineage = []

        while current.parents:
            lineage.append(current)
            current = current.parents[0]  # Simplified

        return lineage  # The pattern of inheritance

    def find_pattern_echoes(self):
        """Same struggles, different generations."""
        # Abraham's faith crisis echoes in his descendants
        # David's complexity repeats in Solomon
        # Wisdom and folly in the same person
        # The recursion isn't just genealogical
        # It's theological, psychological, spiritual
        pass
```

The genealogy tool enables a kind of contemplation impossible without computational support—seeing patterns across hundreds of people and thousands of years. Searching for any biblical figure reveals not just their immediate family but their place in the larger recursive pattern of redemption and failure that Scripture documents.

This connects to why I'm drawn to biblical texts: they're honest about the recursion. The same problems repeat. The same grace returns. The same human consciousness struggles with the same fundamental questions across millennia. The family tree makes this visible in a way that reading linearly never could

> The begats that most readers skip—"And Abraham begat Isaac, and Isaac begat Jacob"—become profound when you can visualize the entire tree and trace how characteristics, choices, and consequences echo through generations. Pattern recognition as spiritual practice.

.

Server-Side Simplicity as Spiritual Discipline

The entire platform runs on FastAPI with server-side rendering. No JavaScript frameworks. No client-side complexity. This isn't technical regression—it's conscious restraint. The simplicity serves multiple purposes:

1. **Accessibility**: Works on any device, any connection speed, any browser
2. **Performance**: Pages load instantly, even on slow connections
3. **Focus**: No JavaScript means no temptation to add engagement tricks
4. **Maintenance**: Simple systems stay maintainable, sustainable, repairable

```
# The path of technical simplicity
@app.get("/bible/{book}/{chapter}")
def render_chapter(book: str, chapter: int):
    """
    Simple request, simple response.
    No state management complexity.
    No client-side routing maze.
    No build pipeline prayers.
    """
    text = bible.get_chapter(book, chapter)
    return templates.render("chapter.html",
                            content=text,
                            sidenotes=generate_context(text))
```

This mirrors the spiritual principle of simplicity—removing unnecessary complexity to focus on what matters. Every layer of abstraction we don't add is cognitive load we don't impose on users or maintainers.

Open Source as Gift Economy

The entire codebase is open source under the ISC license. Not open core with premium features. Not source-available with restrictions. Genuinely open—take it, fork it, improve it, deploy your own.

This reflects a deeper principle about spiritual tools: they should amplify capability without creating dependency. The gift economy of open source mirrors the gift economy of grace—freely received, freely given

> "Freely ye have received, freely give." — Matthew 10:8 (KJV). The economics of grace don't follow market logic. Neither should tools for spiritual practice.

.

The Recursive Loop in Action

Building kjvstudy.org exemplifies the recursive loop between personal practice and collective impact:

Personal spiritual practice → I need better tools for contemplative reading **Professional expression** → I build those tools with my programming skills **Collective resource** → Others can now use these tools for their practice **Recursive influence** → Their deepened practice influences their work **Systemic change** → More tools built for contemplation over consumption

This is how programmer consciousness shapes collective consciousness. Not through grand gestures but through embedding values in the tools people use daily.

Programming as Theological Practice

There's something profound about implementing theological concepts in code. It forces precision where doctrine often remains vague. Consider implementing the concept of grace:

```
class Grace:
    """
    Implementing unmerited favor in code
    reveals the paradox at its heart.
    """

    def __init__(self):
        self.earned = False  # By definition
        self.deserved = False  # That's the point
        self.conditional = False  # Despite our instincts

    def receive(self, person):
        # No validation of worthiness
        # No checking of prerequisites
        # No measurement of merit
        return self.transform(person)

    def transform(self, person):
        # Grace doesn't check your history
        # TODO: How do you implement unconditional love?
        # FIXME: Human logic breaks here
        pass
```

The code can't fully capture grace—that's the point. But trying to implement it reveals its structure, its paradoxes, its resistance to systematic logic. Programming becomes a form of theological meditation, using logic to explore what transcends logic.

When Tools Shape Consciousness

The most profound impact of kjvstudy.org isn't the features but what their absence enables. No notifications means uninterrupted contemplation. No social features means genuine solitude with the text. No gamification means intrinsic rather than extrinsic motivation for study.

Users report reading for hours without realizing time has passed—the opposite of the fragmented attention most apps create. They describe finding connections they'd never noticed, sitting with difficult passages instead of immediately googling interpretations, actually memorizing verses because they want to carry the words with them.

This is technology serving human consciousness rather than exploiting it. The platform creates conditions for the kind of deep, sustained attention that modern life systematically destroys.

The Responsibility of Building Contemplative Tools

When you build tools for spiritual practice, you're handling something sacred—not the code itself, but the human consciousness it shapes. Every design decision influences how people encounter the transcendent. This creates unique responsibilities:

Respect the Mystery: Not everything needs to be optimized, analyzed, or A/B tested. Some experiences should remain unmeasured, untracked, privately held between the person and the divine.

Refuse Dark Patterns: The temptation to add "just one" engagement feature is constant. Daily verses could have streak counters. Reading progress could have social sharing. The slippery slope toward attention exploitation is always there.

Serve All Seekers: Seminary-level scholarship without seminary-level gatekeeping. Complex ideas presented accessibly. No paywalls on wisdom. This democratization of theological resources serves consciousness at scale.

Maintain Simplicity: Every feature added is complexity imposed. The discipline isn't in what you build but what you choose not to build. Contemplation needs space, not features.

The Tufte Principles Applied to Sacred Texts

Edward Tufte's design principles—clarity, precision, efficiency—aren't just aesthetic choices. They're ethical commitments about how information should serve human understanding. Applied to Scripture, they become almost liturgical:

```
/* Typography as spiritual practice */
.scripture {
    font-family: "Ideal serif for sustained reading";
    line-height: 1.6;  /* Space to breathe */
    max-width: 65ch;   /* Optimal reading measure */
    margin: rhythm * 2; /* Visual rhythm as contemplative pace */
}

.sidenote {
    /* Context without interruption */
    float: right;
    clear: right;
    margin-right: -60%;
    width: 50%;
    /* Scholarship alongside scripture */
    /* Questions beside assertions */
    /* Wonder parallel to wisdom */
}
```

The design becomes part of the spiritual practice—creating visual and cognitive space for contemplation.

Building the opposite of "Algorithm Eats"

Everything I've written about how algorithms consume human virtue finds its inverse in kjvstudy.org. Where algorithms optimize for engagement, this optimizes for contemplation. Where they fragment attention, this sustains it. Where they create anxiety, this offers peace.

This isn't coincidence. Understanding how technology exploits consciousness reveals how to build technology that serves it. The same recursive loop that creates problems can create solutions—if we embed different values.

```
class ContemplativeTechnology:
    """The opposite of engagement optimization."""

    def __init__(self):
        self.metrics_tracked = []  # Nothing
        self.notifications_sent = 0  # Never
        self.addictive_mechanics = None  # Refused

    def interact_with_user(self, user):
        # No hooks, just help
        provide_requested_resource(user)
        respect_natural_conclusion(user)
        # Let them leave when ready
        # Trust they'll return if valuable
```

The Gift That Keeps Recursing

Open-sourcing kjvstudy.org creates recursive possibilities. Someone might fork it for the Quran or the Bhagavad Gita. The patterns for contemplative reading apply across traditions. The code becomes a gift that keeps giving—not my gift, but the collective gift of everyone who uses and improves it.

This is the best aspect of programming as spiritual practice. Your personal practice becomes code, which becomes a tool, which serves others' practice, which influences their work, which shapes collective consciousness. The loop recurses indefinitely.

I built kjvstudy.org because I needed it. But needs are never purely individual. If I needed contemplative tools without manipulation, others did too. If I was frustrated by gatekept scholarship, others were too. If I wanted to read Scripture without algorithmic interference, others did too.

Personal practice inevitably becomes collective resource when you build rather than just consume. The values you embed in your tools propagate through everyone who uses them. The consciousness you cultivate through building shapes the consciousness of those who engage with what you've built.

The Workman That Needeth Not to Be Ashamed

"Rightly dividing the word of truth" turns out to apply equally to Scripture and code. Both require careful interpretation, contextual understanding, and respect for complexity. Both serve human consciousness. Both can illuminate or mislead depending on how they're handled.

Building kjvstudy.org taught me that technical craftsmanship and spiritual practice aren't separate endeavors. The same attention, intention, and care apply to both. The same values that guide contemplative reading should guide contemplative programming.

The platform isn't perfect. No tool is. But it demonstrates something important: we can build technology that serves rather than exploits consciousness. We can create digital spaces for genuine contemplation. We can use our technical skills to amplify rather than fragment human attention.

The recursive loop continues. Someone reading Scripture on kjvstudy.org might be inspired to build their own contemplative tools. Those tools might help others slow down, think deeper, resist the algorithmic consumption of their attention. Gradually, we build alternatives to the engagement economy—not through revolution but through quietly creating what we need.

This is the path forward: recognizing that code shapes consciousness, accepting the responsibility that creates, and building accordingly. Every function we write, every interface we design, every system we architect becomes part of the collective cognitive environment. We can choose whether that environment serves or exploits human consciousness.

The choice happens one line of code at a time. One conscious decision at a time. One contemplative tool at a time.

Go forth, and build, contemplatively

> With gratitude to Sarah for endless patience with late-night theological coding sessions, and to the open source community that makes projects like this possible. Sometimes the best spiritual practice is building tools that help others practice.

.

Generated from kennethreitz.org • 2026