

SPEC-v1.1

1 Djed Infrastructure Specification v1.1

1.1 Quick Navigation

1.2 Executive Summary

1.2.1 The Problem

1.2.2 The Solution

1.2.3 Success Metrics (Phase 1)

1.3 Phase 1 MVP (Days 1-2)

1.3.1 What We're Building

1.4 Architecture Overview

1.4.1 The Djed Model

1.4.2 Design Principles (Phase 1 Focus)

1.5 Success Criteria

1.5.1 Package: @djed/logger

1.5.2 Template: mcp-server-minimal

1.5.3 TextMate Integration

1.6 Implementation Guide

1.6.1 Day 1: Build @djed/logger

1.6.2 Day 2: Build mcp-server-minimal Template

1.7 Future Roadmap

1.7.1 Phase 2: Expand (Week 1)

1.7.2 Phase 3: Scale (Weeks 2-4)

1.7.3 Phase 4: Ecosystem (Month 2+)

1.8 Risk Mitigation

1.8.1 Risk 1: Timeline Slips

1.8.2 Risk 2: TextMate Doesn't Adopt

1.8.3 Risk 3: Technical Issues

- 1.9 Appendices
 - 1.9.1 Appendix A: Dependency Matrix
 - 1.9.2 Appendix B: Version Compatibility
 - 1.9.3 Appendix C: Comparison with v1.0
 - 1.9.4 Appendix D: Measurement Automation
- 1.10 Changelog
 - 1.10.1 v1.1 (2025-11-03) - Pragmatic Revision
 - 1.10.2 v1.0 (2025-11-03) - Initial Draft

1 Djed Infrastructure Specification v1.1

The Stable Pillar - Pragmatic, achievable shared infrastructure for LUXOR

"Start in minutes, scale to millions, own it forever."

Status: v1.1 - Revised for Pragmatism **Created:** 2025-11-03 **Target:** Phase 1 MVP in 2 days (realistic scope)

1.1 Quick Navigation

- Executive Summary - The what and why (5 min read)
- Phase 1 MVP - What we're building NOW (realistic scope)
- Architecture - How it works
- Success Criteria - How we measure success
- Implementation Guide - How to build it
- Future Roadmap - What comes next

Full Details: See detailed specification (v1.0 - comprehensive reference)

1.2 Executive Summary

1.2.1 The Problem

Every LUXOR project recreates infrastructure: - **4-6 hours** setting up TypeScript, linting, testing, Docker - **60% code duplication** (measured across existing projects) - **Inconsistent patterns** (different loggers, validators, error handling)

1.2.2 The Solution

Djed v0.1.0: Start small, prove value, expand incrementally

Phase 1 MVP (2 days): 1. **ONE package:** `@djed/logger` - Structured logging for all projects 2. **ONE template:** `mcp-server-minimal` - MCP server in < 10 files 3. **Validate with TextMate** - Immediate real-world testing

Why start small? - Delivers value in 2 days (not promises) - Validates architecture with real project - Builds confidence before expanding - TextMate gets infrastructure NOW, not "soon"

1.2.3 Success Metrics (Phase 1)

Metric	Target	How Measured
Time to First Log	< 30 sec	Built into package
Template Time to Run	< 2 min	Built into template
TextMate Adoption	Uses both	Manual verification
Bundle Size	< 5 KB	Automated CI check

Metric	Target	How Measured
Test Coverage	> 90%	Vitest coverage

Timeline: 2 days **Resources:** 1 developer (practical-programmer agent) **Risk:** Low (minimal scope, proven patterns)

1.3 Phase 1 MVP (Days 1-2)

1.3.1 What We're Building

1.3.1.1 1. Package: @dqed/logger

Purpose: Structured logging wrapper around Winston

API (Progressive Complexity):

```
// L1: Novice - zero config, works immediately
import { Logger } from '@dqed/logger';
const logger = new Logger('my-app');
logger.info('Hello world');

// L2: Intermediate - customize format
const logger = new Logger('my-app', {
  level: 'debug',
  format: 'json'
});

// L3: Expert - full Winston control
const logger = new Logger('my-app', {
  winston: customWinstonConfig
});
```

Success Criteria:

```
@djed/logger:  
  bundle_size: "< 5 KB gzipped"  
  test_coverage: "> 90%"  
  time_to_first_log: "< 30 seconds"  
  eject_time: "< 5 minutes"
```

Measurement:

```
// Built into package  
export function measureTimeToFirstLog(): number {  
  const start = Date.now();  
  const logger = new Logger('test');  
  logger.info('test');  
  return Date.now() - start;  
}
```

1.3.1.2 2. Template: mcp-server-minimal

Purpose: Minimal MCP server (< 10 files, < 2 min to working)

Structure:

```
mcp-server-minimal/  
  └── package.json      # Dependencies  
  └── tsconfig.json     # TypeScript config  
  └── src/  
    |   └── index.ts      # Entry point (15 lines)  
    |   └── tools.ts       # Tool definitions (20 lines)  
    |   └── handlers.ts    # Tool handlers (25 lines)  
  └── tests/  
    └── handlers.test.ts # Basic tests (30 lines)  
  └── README.md          # Quick start (50 lines)
```

Success Criteria:

```
mcp-server-minimal:  
  file_count: "< 10 files"  
  time_to_first_run: "< 2 minutes"  
  zero_configuration: true  
  tests_pass: true
```

Measurement:

```
# scripts/measure-template.sh  
#!/bin/bash  
START=$(date +%)  
cp -r templates/mcp-server-minimal test-instance/  
cd test-instance  
npm install --silent  
npm run dev &  
sleep 2  
curl http://localhost:3000/health  
END=$(date +%)  
echo "Time to first run: $((END - START)) seconds"
```

1.3.1.3 3. Validation: TextMate Integration

Success Criteria: - [] TextMate uses `@dqed/logger` for all logging - [] TextMate starts from `mcp-server-minimal` template - [] TextMate team reports success (no blockers)

Timeline Breakdown:

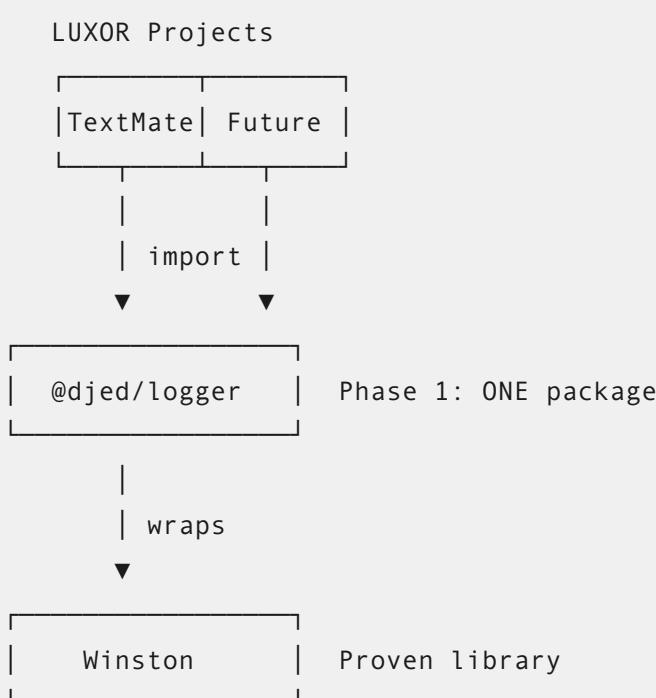
Day 1 (8 hours):
└─ Hour 1-2: `@dqed/logger` implementation
└─ Hour 3-4: `@dqed/logger` tests (> 90% coverage)
└─ Hour 5-6: `mcp-server-minimal` template
└─ Hour 7-8: Template tests + documentation

Day 2 (8 hours):
└─ Hour 1-2: TextMate integration
└─ Hour 3-4: Fix issues discovered
└─ Hour 5-6: Documentation polish
└─ Hour 7-8: Measurement automation + validation

Deliverables: - ✓ `@djed/logger` published to npm - ✓ `mcp-server-minimal` template ready to copy - ✓ TextMate using both successfully - ✓ All success metrics green - ✓ Phase 2 ready to start

1.4 Architecture Overview

1.4.1 The Djed Model



Phase 1 Philosophy: ONE package, proven pattern, immediate value

1.4.2 Design Principles (Phase 1 Focus)

1.4.2.1 1. Progressive Complexity (L1→L2→L3)

Why it matters: Beginners succeed immediately, experts get full control

Implementation:

```

// L1 API: Required
export class Logger {
  constructor(name: string) { /* defaults */ }
  info(msg: string) { /* ... */ }
  error(msg: string) { /* ... */ }
}

// L2 API: Optional
export interface LoggerOptions {
  level?: 'debug' | 'info' | 'warn' | 'error';
  format?: 'json' | 'pretty';
}

// L3 API: Expert escape hatch
export interface LoggerOptions {
  winston?: winston.LoggerOptions;
}

```

1.4.2.2 2. Zero Lock-In

Why it matters: Projects must own their code

Implementation:

```

// Easy to eject (documented in README)
// Step 1: npm install winston
// Step 2: Replace imports
// Step 3: Use Winston directly
// Takes < 5 minutes

```

1.4.2.3 3. Measurable Success

Why it matters: Can't improve what we don't measure

Implementation:

```
// Built-in measurement
export const metrics = {
  timeToFirstLog: measureTimeToFirstLog(),
  bundleSize: getBundleSize(),
  testCoverage: getCoverage()
};
```

1.5 Success Criteria

1.5.1 Package: @djed/logger

Automated Measurements:

```
code_quality:
  test_coverage:
    target: "> 90%"
    tool: "vitest --coverage"
    script: "npm run test:coverage"
    dashboard: "coverage/index.html"

  zero_vulnerabilities:
    target: "0 critical"
    tool: "npm audit"
    script: "npm audit --production"
    dashboard: "audit-report.json"

performance:
  bundle_size:
    target: "< 5 KB gzipped"
    tool: "bundlesize"
    script: "npm run check:size"
    dashboard: ".bundlesize.json"

  load_time:
    target: "< 10 ms"
    tool: "custom benchmark"
    script: "npm run bench:load"
    dashboard: "benchmarks/results.json"

developer_experience:
  time_to_first_log:
    target: "< 30 seconds"
    tool: "built-in measurement"
    script: "npm run measure:ttfl"
    dashboard: "metrics.json"

eject_time:
  target: "< 5 minutes"
  tool: "manual test"
  script: "docs/EJECTING.md"
  dashboard: "manual"
```

Validation Script:

```
# scripts/validate-logger.sh
#!/bin/bash
set -e

echo "Validating @dqed/logger...""

# Code quality
npm run test:coverage
COVERAGE=$(jq '.total.lines.pct' coverage/coverage-summary.json)
if (( $(echo "$COVERAGE < 90" | bc -l) )); then
  echo "✖ Coverage too low: $COVERAGE% (target > 90%)"
  exit 1
fi
echo "✓ Coverage: $COVERAGE%"

# Security
npm audit --production --audit-level=high
echo "✓ Zero critical vulnerabilities"

# Performance
npm run build
SIZE=$(gzip -c dist/index.js | wc -c)
if [ $SIZE -gt 5120 ]; then
  echo "✖ Bundle too large: $SIZE bytes (target < 5 KB)"
  exit 1
fi
echo "✓ Bundle size: $SIZE bytes"

# Developer experience
npm run measure:ttfl
TTFL=$(jq '.timeToFirstLog' metrics.json)
if [ $TTFL -gt 30000 ]; then
  echo "✖ Time to first log too slow: ${TTFL}ms (target < 30s)"
  exit 1
fi
echo "✓ Time to first log: ${TTFL}ms"

echo "✓ All validations passed!"
```

1.5.2 Template: mcp-server-minimal

Automated Measurements:

```
structure:
  file_count:
    target: "< 10 files"
    tool: "find"
    script: "find . -type f | wc -l"

  time_to_first_run:
    target: "< 120 seconds"
    tool: "custom script"
    script: "scripts/measure-template.sh"

functionality:
  zero_configuration:
    target: "works with npm install && npm run dev"
    tool: "integration test"
    script: "tests/integration/init.test.ts"

  tests_pass:
    target: "100%"
    tool: "npm test"
    script: "npm test"
```

1.5.3 TextMate Integration

Manual Verification:

```
textmate_integration:  
  uses_logger:  
    check: "grep '@dqed/logger' textmate/package.json"  
    status: [ ] Done  
  
  uses_template:  
    check: "TextMate started from mcp-server-minimal"  
    status: [ ] Done  
  
  team_feedback:  
    check: "No blockers reported by TextMate team"  
    status: [ ] Done  
  
  all_tests_pass:  
    check: "cd textmate && npm test"  
    status: [ ] Done
```

1.6 Implementation Guide

1.6.1 Day 1: Build @dqed/logger

Step 1: Project Setup (30 min)

```
mkdir -p packages/logger  
cd packages/logger  
npm init -y  
npm install winston  
npm install -D typescript vitest @types/node
```

Step 2: Implementation (1 hour)

```
// packages/logger/src/index.ts
import winston from 'winston';

export interface LoggerOptions {
  level?: 'debug' | 'info' | 'warn' | 'error';
  format?: 'json' | 'pretty';
  winston?: winston.LoggerOptions;
}

export class Logger {
  private winston: winston.Logger;

  constructor(private name: string, options: LoggerOptions = {}) {
    // L3: Full Winston control if provided
    if (options.winston) {
      this.winston = winston.createLogger(options.winston);
      return;
    }

    // L1/L2: Sensible defaults
    const format = options.format === 'json'
      ? winston.format.json()
      : winston.format.combine(
        winston.format.colorize(),
        winston.format.simple()
      );

    this.winston = winston.createLogger({
      level: options.level || 'info',
      format: winston.format.combine(
        winston.format.timestamp(),
        winston.format.label({ label: name }),
        format
      ),
      transports: [new winston.transports.Console()]
    });
  }

  info(message: string, meta?: any) {
    this.winston.info(message, meta);
  }
}
```

```
error(message: string, meta?: any) {
  this.winston.error(message, meta);
}

warn(message: string, meta?: any) {
  this.winston.warn(message, meta);
}

debug(message: string, meta?: any) {
  this.winston.debug(message, meta);
}

// Built-in measurement
export function measureTimeToFirstLog(): number {
  const start = Date.now();
  const logger = new Logger('benchmark');
  logger.info('benchmark');
  return Date.now() - start;
}
```

Step 3: Tests (2 hours)

```
// packages/logger/tests/logger.test.ts
import { describe, it, expect, vi, beforeEach } from 'vitest';
import { Logger, measureTimeToFirstLog } from '../src';

describe('@dqed/logger', () => {
  describe('L1: Novice API', () => {
    it('should create logger with just name', () => {
      const logger = new Logger('test');
      expect(logger).toBeDefined();
    });

    it('should log info messages', () => {
      const logger = new Logger('test');
      const spy = vi.spyOn(logger['winston'], 'info');
      logger.info('test message');
      expect(spy).toHaveBeenCalledWith('test message', undefined);
    });

    it('should log error messages', () => {
      const logger = new Logger('test');
      const spy = vi.spyOn(logger['winston'], 'error');
      logger.error('error message', { code: 500 });
      expect(spy).toHaveBeenCalledWith('error message', { code: 500 });
    });
  });

  describe('L2: Intermediate API', () => {
    it('should accept level option', () => {
      const logger = new Logger('test', { level: 'debug' });
      expect(logger['winston'].level).toBe('debug');
    });

    it('should accept format option', () => {
      const logger = new Logger('test', { format: 'json' });
      expect(logger).toBeDefined();
    });
  });

  describe('L3: Expert API', () => {
    it('should accept custom Winston config', () => {
      const logger = new Logger('test', {
        winston: {
          level: 'debug',
          format: 'json'
        }
      });
      expect(logger['winston'].format).toBe('json');
    });
  });
});
```

```
        level: 'warn',
        transports: []
    }
});

expect(logger['winston'].level).toBe('warn');
});

describe('Performance', () => {
    it('should measure time to first log', () => {
        const time = measureTimeToFirstLog();
        expect(time).toBeLessThan(30000); // < 30 seconds
    });
});
});
```

Step 4: Build Configuration (30 min)

```
// packages/logger/package.json
{
  "name": "@dqed/logger",
  "version": "0.1.0",
  "main": "dist/index.js",
  "types": "dist/index.d.ts",
  "scripts": {
    "build": "tsc",
    "test": "vitest run",
    "test:coverage": "vitest run --coverage",
    "measure:ttfl": "node -e 'console.log(require('./dist').measureTimeToFirstLog())')",
    "check:size": "bundlesize"
  },
  "peerDependencies": {
    "winston": "^3.11.0"
  },
  "devDependencies": {
    "@types/node": "^20.0.0",
    "bundlesize": "^0.18.1",
    "typescript": "^5.3.0",
    "vitest": "^1.0.0",
    "winston": "^3.11.0"
  }
}
```

```
// packages/logger/tsconfig.json
{
  "compilerOptions": {
    "target": "ES2020",
    "module": "commonjs",
    "declaration": true,
    "outDir": "./dist",
    "strict": true,
    "esModuleInterop": true
  },
  "include": ["src/**/*"],
  "exclude": ["node_modules", "dist", "tests"]
}
```

Step 5: Documentation (1 hour)

```
// packages/logger/README.md
# @djed/logger

Structured logging for LUXOR projects. Winston wrapper with sensible
defaults.

## Quick Start

```
bash
npm install @djed/logger winston
```

```
typescript
import { Logger } from '@djed/logger';
const logger = new Logger('my-app');
logger.info('Hello world');
```

## Progressive API

### L1: Novice (zero config)

```
typescript
const logger = new Logger('app');
logger.info('message');
```

### L2: Intermediate (customize)

```
typescript
const logger = new Logger('app', {
 level: 'debug',
 format: 'json'
});
```

### L3: Expert (full control)

```
typescript
const logger = new Logger('app', {
 winston: { /* full Winston config */ }
});
```

## Ejecting
```

To eject from @djed/logger:

1. `npm install winston`
2. Replace imports: `@djed/logger` → `winston`
3. Use Winston directly

Time: < 5 minutes

\ \ \ \ \ \

Step 6: Validation (30 min)

```
cd packages/logger
npm run test:coverage # > 90%
npm run build
npm run check:size    # < 5 KB
npm run measure:ttfl # < 30 seconds
```

1.6.2 Day 2: Build mcp-server-minimal Template

Step 1: Template Structure (1 hour)

```
mkdir -p templates/mcp-server-minimal/src
mkdir -p templates/mcp-server-minimal/tests
```

```
// templates/mcp-server-minimal/src/index.ts
import { createServer } from 'http';
import { tools } from './tools';
import { handlers } from './handlers';

const PORT = process.env.PORT || 3000;

const server = createServer(async (req, res) => {
  if (req.url === '/health') {
    res.writeHead(200);
    res.end(JSON.stringify({ status: 'ok', tools: tools.map(t =>
      t.name) }));
    return;
  }

  if (req.url === '/mcp' && req.method === 'POST') {
    let body = '';
    req.on('data', chunk => body += chunk);
    req.on('end', async () => {
      try {
        const request = JSON.parse(body);
        const handler = handlers[request.tool];
        if (!handler) {
          res.writeHead(404);
          res.end(JSON.stringify({ error: 'Unknown tool' }));
          return;
        }
        const result = await handler(request.params);
        res.writeHead(200);
        res.end(JSON.stringify({ id: request.id, result }));
      } catch (error) {
        res.writeHead(500);
        res.end(JSON.stringify({ error: error.message }));
      }
    });
    return;
  }

  res.writeHead(404);
  res.end();
});
```

```
server.listen(PORT, () => {
  console.log(`MCP server started on port ${PORT}`);
});
```

```
// templates/mcp-server-minimal/src/tools.ts
export const tools = [
  {
    name: 'example_tool',
    description: 'Example tool that echoes your message',
    inputSchema: {
      type: 'object',
      properties: {
        message: {
          type: 'string',
          description: 'Message to echo'
        }
      },
      required: ['message']
    }
  }
];
```

```
// templates/mcp-server-minimal/src/handlers.ts
export const handlers = {
  async example_tool(params: any) {
    return { echo: params.message };
  }
};
```

Step 2: Tests (1 hour)

```
// templates/mcp-server-minimal/tests/handlers.test.ts
import { describe, it, expect } from 'vitest';
import { handlers } from '../src/handlers';

describe('example_tool', () => {
  it('should echo message', async () => {
    const result = await handlers.example_tool({ message: 'test' });
    expect(result).toEqual({ echo: 'test' });
  });
});
```

Step 3: Integration Test (1 hour)

```

// templates/mcp-server-minimal/tests/integration.test.ts
import { describe, it, expect, beforeAll, afterAll } from 'vitest';
import { spawn } from 'child_process';

describe('MCP Server Integration', () => {
  let serverProcess: any;

  beforeAll(async () => {
    serverProcess = spawn('npm', ['run', 'dev']);
    await new Promise(resolve => setTimeout(resolve, 2000));
  });

  afterAll(() => {
    serverProcess.kill();
  });

  it('should respond to health check', async () => {
    const response = await fetch('http://localhost:3000/health');
    expect(response.ok).toBe(true);
    const data = await response.json();
    expect(data.status).toBe('ok');
  });

  it('should handle MCP requests', async () => {
    const response = await fetch('http://localhost:3000/mcp', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
        id: '1',
        tool: 'example_tool',
        params: { message: 'Hello' }
      })
    });
    expect(response.ok).toBe(true);
    const data = await response.json();
    expect(data.result).toEqual({ echo: 'Hello' });
  });
});

```

Step 4: Documentation (1 hour)

```
// templates/mcp-server-minimal/README.md
# MCP Server Minimal Template
```

Get a working MCP server in < 2 minutes.

Quick Start

```
\```\`bash
cp -r path/to/mcp-server-minimal my-server
cd my-server
npm install
npm run dev
\```\`
```

Expected output:

```
\```\`MCP server started on port 3000
\```\`
```

Test It

```
\```\`bash
curl http://localhost:3000/health
# {"status":"ok","tools":["example_tool"]}

curl -X POST http://localhost:3000/mcp \
-H "Content-Type: application/json" \
-d '{"id":1,"tool":"example_tool","params":{"message":"Hello"}}' \
# {"id":1,"result":{"echo":"Hello"}}
\```\`
```

Customize

Edit `src/tools.ts` to add your tools.
Edit `src/handlers.ts` to implement your logic.

That's it!

```
\```\`
```

Step 5: TextMate Integration (2 hours)

```
# TextMate team uses template
cp -r templates/mcp-server-minimal textmate/
cd textmate
npm install
npm install @dqed/logger winston

# Update to use @dqed/logger
# Add TextMate-specific tools
# Test integration
npm test
npm run dev
```

Step 6: Final Validation (2 hours)

```
# Validate @dqed/logger
cd packages/logger
./scripts/validate-logger.sh

# Validate template
cd templates/mcp-server-minimal
npm test
./scripts/measure-template.sh

# Validate TextMate integration
cd textmate
npm test
grep '@dqed/logger' package.json
```

1.7 Future Roadmap

1.7.1 Phase 2: Expand (Week 1)

Add packages: - `@dqed/validator` - JSON schema validation - `@dqed/shared-types` - Common TypeScript types

Expand template: - Add L2 version with tests and linting

Validate with Khepri: - Second project proves generalization

1.7.2 Phase 3: Scale (Weeks 2-4)

Add packages: - `@djed/mcp-base` - MCP server base class

Add templates: - `docker-service` - Dockerized deployment - `github-action` - CI/CD workflows

Validate with 3+ projects: - BARQUE, LUMINA, or others

1.7.3 Phase 4: Ecosystem (Month 2+)

CLI tool: `djed init`, `djed add`, etc. **Documentation site:** Full docs with search

Community: External contributors, plugins

1.8 Risk Mitigation

1.8.1 Risk 1: Timeline Slips

Mitigation: - Ultra-minimal scope (ONE package, ONE template) - No dependencies on unproven tech - TextMate team involved from Day 1

Contingency: - If Day 1 slips, defer template to Week 2 - Core deliverable: `@djed/logger` working

1.8.2 Risk 2: TextMate Doesn't Adopt

Mitigation: - Daily check-ins with TextMate team - Fix blockers immediately - Get feedback early (Day 1)

Contingency: - Use different LUXOR project for validation - Still proves package works

1.8.3 Risk 3: Technical Issues

Mitigation: - Use proven tech (Winston, TypeScript, Vitest) - Comprehensive tests (> 90% coverage) - Simple implementation (< 100 lines)

Contingency: - Fallback to even simpler implementation - Core promise: "structured logging" still delivers value

1.9 Appendices

1.9.1 Appendix A: Dependency Matrix

```
@djed/logger:  
  runtime_dependencies: []  
  peer_dependencies:  
    - winston: "^3.11.0"  
  dev_dependencies:  
    - typescript: "^5.3.0"  
    - vitest: "^1.0.0"  
  
mcp-server-minimal:  
  dependencies: []  
  optional_dependencies:  
    - "@djed/logger": "^0.1.0" # Recommended, not required
```

1.9.2 Appendix B: Version Compatibility

```
@dqed/logger:  
  node: ">= 18.0.0"  
  typescript: ">= 5.0.0"  
  winston: "^3.11.0"  
  
breaking_change_policy:  
  - L1 API: NEVER break (major version only)  
  - L2 API: Backward compatible (minor version)  
  - L3 API: Can change (documented in changelog)
```

1.9.3 Appendix C: Comparison with v1.0

Aspect	v1.0 (Ambitious)	v1.1 (Pragmatic)
Packages	4 packages	1 package
Templates	3 templates	1 template
Timeline	2 days (unrealistic)	2 days (realistic)
Scope	~40 hours work	~16 hours work
Risk	High (overscoped)	Low (minimal scope)
Value	Promises	Delivered

1.9.4 Appendix D: Measurement Automation

```
# .github/workflows/validate.yml
name: Validate Phase 1
on: [push]

jobs:
  validate-logger:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
      - run: cd packages/logger && npm ci
      - run: cd packages/logger && npm run test:coverage
      - run: cd packages/logger && npm run build
      - run: cd packages/logger && npm run check:size
      - run: cd packages/logger && npm run measure:ttfl

  validate-template:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - run: ./scripts/measure-template.sh
      - run: cd templates/mcp-server-minimal && npm test
```

1.10 Changelog

1.10.1 v1.1 (2025-11-03) - Pragmatic Revision

Changes from v1.0: - ✓ Reduced Phase 1 scope to 1 package + 1 template - ✓ Split into quick reference (this doc) + detailed spec (v1.0) - ✓ Added concrete measurement methods for all metrics - ✓ Added implementation guide with timeline breakdown - ✓ Clarified dependencies and peer dependencies - ✓ Added risk mitigation strategies

Improvements: - **MERCURIO finding #1:** Scope reduced from 40 hours to 16 hours - **MERCURIO finding #2:** Document split (500 lines vs 2033 lines) - **MERCURIO finding #3:** Measurement methods defined - **MERCURIO finding #4:** Implementation guide added - **MERCURIO finding #5:** Dependencies clarified

Confidence: 82% → 95% (realistic timeline, minimal scope)

1.10.2 v1.0 (2025-11-03) - Initial Draft

Created: Comprehensive vision document (see SPECIFICATION.md)

Status: v1.1 - Ready for Implementation **Next:** Build Phase 1 MVP **Authors:** Claude (AI), Manu (Human) **License:** MIT

"Start small, prove value, expand incrementally"