Design Standard

Directives and Guidelines
Version 1.0

Author: Juval Löwy

www.idesign.net

Table of Content

Preface

The system design and project design ideas that I write about and teach are simple and consistent both internally and with every other engineering discipline. However, it can be overwhelming at first to come to terms with this new way of thinking. Over time and with practice, applying these concepts becomes second nature. To facilitate absorbing them all, this document offers a concise design standard as a checklist. The standard contains all the best practices from *Righting Software* and the two Master Classes (the *Architect's Master Class* and the *Project Design Master Class*), ideas we have practiced successfully at IDesign across countless projects.

The list of practices on its own will not mean much because you still must know the context for each item in the list. Nevertheless, referring to the standard can ensure that you do not omit an important attribute or consideration. This makes the standard essential for successful system and project design delivery by helping you enforce the best practices and avoid the pitfalls.

The standard contains two types of items: directives and guidelines. A **directive** is a rule that you should never violate, since doing so is certain to cause the project to fail. A **guideline** is a piece of advice that you should follow unless you have a strong and unusual justification for going against it. Violating a guideline alone is not certain to cause the project to fail, but too many violations will tip the project into failure. It is also unlikely that if you abide by the directives that you will have any reason to go against the guidelines.

Juval Löwy

The Prime Directive

Never design against the requirements.

Directives

1. Avoid functional decomposition.
2. Decompose based on volatility.
3. Provide a composable design.
4. Offer features as aspects of integration, not implementation.
5. Design iteratively, build incrementally.
6. Design the project to build the system.
7. Drive educated decisions with viable options that differ by schedule, cost, and risk.
8. Build the project along its critical path.
9. Be on time throughout the project.

System Design Guidelines

1. **Requirements**
   a. Capture required behavior, not required functionality.
   b. Describe required behavior with use cases.
   c. Document all use cases that contain nested conditions with activity diagrams.
   d. Eliminate solutions masquerading as requirements.
   e. Validate the system design by ensuring it supports all core use cases.
2. **Cardinality**
   a. Avoid more than five *Managers* in a system without subsystems.
   b. Avoid more than a handful of subsystems.
   c. Avoid more than three *Managers* per subsystem.
   d. Strive for a golden ratio of *Engines* to *Managers*.
   e. Allow *ResourceAccess* components to access more than one *Resource* if necessary.

3. **Attributes**
   a. Volatility should decrease top-down.
   b. Reuse should increase top-down.
   c. Do not encapsulate changes to the nature of the business.
   d. *Managers* should be almost expendable.
   e. Design should be symmetric.
   f. Never use a public communication channels for internal system interactions.
4. **Layers**
   a. Avoid open architecture.
   b. Avoid semi-closed/semi-open architecture.
   c. Prefer a closed architecture.
      i) Do not call up.
      ii) Do not call sideways (except queued calls between *Managers*).
      iii) Do not call more than one layer down.
      iv) Resolve attempts at opening the architecture by using queued calls or asynchronous event publishing.
   d. Extend the system by implementing subsystems.
5. **Interaction rules**
   a. All components can call *Utilities*.
   b. *Managers* and *Engines* can call *ResourceAccess*.
   c. *Managers* can call *Engines*.
   d. *Managers* can queue calls to another *Manager.*
6. **Interaction don’ts**
   a. *Clients* do not call multiple *Managers* in the same use case.
   b. *Managers* do not queue calls to more than one *Manager* in the same use case.
   c. *Engines* do not receive queued calls.
   d. *ResourceAccess* components do not receive queued calls.
   e. *Clients* do not publish events.
   f. *Engines* do not publish events.
   g. *ResourceAccess* components do not publish events.
   h. *Resources* do not publish events.
   i. *Engines*, *ResourceAccess*, and *Resources* do not subscribe to events.

Project Design Guidelines

1. **General**
   a. Do not design a clock.
   b. Never design a project without an architecture that encapsulates the volatilities.
   c. Capture and verify planning assumptions.
   d. Follow the design of project design.
   e. Design several options for the project; at a minimum, design normal, compressed, and subcritical solutions.
   f. Communicate with management in Optionality.
   g. Always go through SDP review before the main work starts.
2. **Staffing**
   a. Avoid multiple architects.
   b. Have a core team in place at the beginning.
   c. Ask for only the lowest level of staffing required to progress unimpeded along the critical path.
   d. Always assign resources based on float.
   e. Ensure correct staffing distribution.
   f. Ensure a shallow S curve for the planned earned value.
   g. Always assign components to developers in a 1:1 ratio.
   h. Strive for task continuity.
3. **Integration**
   a. Avoid mass integration points.
   b. Avoid integration at the end of the project.
4. **Estimations**
   a. Do not overestimate.
   b. Do not underestimate.
   c. Strive for accuracy, not precision.
   d. Always use a quantum of five days in any activity estimation.
   e. Estimate the project as a whole to validate or even initiate your project design.
   f. Reduce estimation uncertainty.
   g. When required, maintain correct estimation dialog.

5. **Project network**
   a. Treat resource dependencies as dependencies.
   b. Verify all activities reside on a chain that starts and ends on a critical path.
   c. Verify all activities have a resource assigned to them.
   d. Avoid node diagrams.
   e. Prefer arrow diagrams.
   f. Avoid god activities.
   g. Break large projects into a network of networks.
   h. Treat near-critical chains as critical chains.
   i. Strive for cyclomatic complexity as low as 10 to 12.
   j. Design by layers to reduce complexity.
6. **Time and cost**
   a. Accelerate the project first by quick and clean practices rather than compression.
   b. Never commit to a project in the death zone.
   c. Compress with parallel work rather than top resources.
   d. Compress with top resources carefully and judiciously.
   e. Avoid compression higher than 30%.
   f. Avoid projects with efficiency higher than 25%.
   g. Compress the project even if the likelihood of pursuing any of the compressed options is low.
7. **Risk**
   a. Customize the ranges of criticality risk to your project.
   b. Adjust floats outliers with activity risk.
   c. Decompress the normal solution past the tipping point on the risk curve.
      i) Target decompression to 0.5 risk.
      ii) Value the risk tipping point more than a specific risk number.
   d. Do not over-decompress.
   e. Decompress design-by-layers solutions, perhaps aggressively so.
   f. Keep normal solutions at less than 0.7 risk.
   g. Avoid risk lower than 0.3.
   h. Avoid risk higher than 0.75.
   i. Avoid project options risker or safer than the risk crossover points.

Project Tracking Guidelines

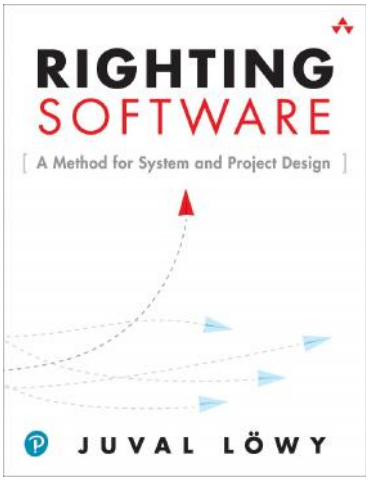
1. Adopt binary exit criteria for internal phases of an activity.
2. Assign consistent phase weights across all activities.
3. Track progress and effort on a weekly basis.
4. Never base progress reports on features.
5. Always base your progress reports on integration points.
6. Track the float of near-critical chains.

Service Contract Design Guidelines

1. Design reusable service contracts.
2. Comply with service contract design metrics
   a. Avoid contracts with a single operation.
   b. Strive to have 3 to 5 operations per service contract.
   c. Avoid service contracts with more than 12 operations.
   d. Reject service contracts with 20 or more operations.
3. Avoid property-like operations.
4. Limit the number of contracts per service to 1 or 2.
5. Avoid junior hand-offs.
6. Have only the architect or competent senior developers design the contracts.

Resources

1. **Righting Software, 1st Edition**



By Juval Löwy, Addison-Wesley, 2020

2. **The Architect's Master Class**

The *Architect's Master Class* is the ultimate resource for the professional architect. The class shows how to take an active leadership role and is often referred to as a career-changing event. Alumni of the class are the architects of some of the most well-known companies and projects around the world. While the class shows how to design modern systems, it sets the focus on the 'why' and the rationale behind particular design decisions, often shedding light on poorly understood aspects. You will see relevant design guidelines, best practices, pitfalls, and the crucial process required of today's modern architects. Don't miss on this unique opportunity to learn and improve your design skills with IDesign, and share our passion for architecture and software engineering.

3. **The Project Design Master Class**

The *Project Design Master Class* presents our structured approach to project design and a comprehensive set of matching tools and techniques. You will master the steps, the interactions, the dynamics, the accurate modeling, the complexity reductions, the metrics, the rationale behind the intuition and experience. You will see how to perfect communication with top management, restore trust, and greatly increase your chance of success. The class provides guidance and knowledge that would otherwise take decades and many projects to acquire, and will propel your career like nothing else ever will.