Software Architecture Fundamentals

Jeff Doolittle, Software Architect and Mentor - https://jeffdoolittle.com

Executive Summary

In the midst of the rush to deliver features and meet aggressive deadlines, almost all projects begin with the best of intentions and an early sense of velocity and productivity. However most projects reach a point where **productivity slows to a crawl**. The **complexity** of the system becomes difficult if not **impossible to manage** efficiently and effectively. A lack of adequate documentation leaves **tacit knowledge in the heads of developers**. Changes in requirements necessitate **major changes to the system**, often leading to **new defects and bugs**. **Maintenance and support headaches** grow and even accelerate over time. **Cost and schedule overruns** become the norm, and **quality continues to degrade**.

Most software development teams lack **lightweight processes and techniques** which can address these significant problems. System Design and Project Design define such lightweight processes and techniques. These are not silver bullets, but engineering disciplines that can provide the following benefits:

- **Consistent delivery on time and on budget with high quality**
- Empowered development teams **making and keeping their commitments** to deliver while experiencing a **huge boost in morale and confidence**
- Project Management **visibility and control over the status and schedule** of the project
- **Rapid delivery of new functionality** as defined and prioritized by Product Owners
- Ability to **accommodate changes in requirements** without massive rework
- Measurement of system **cost and performance** from the beginning of a project, rather than being surprised once it's too late

System Design and Project Design *work together to enable software development teams to* ***Save Time, Save Money and Assure Quality*** *at every step of the Software Development Lifecycle.*

Contents

Critical Considerations

System Design and Project Design provide engineering rigor for software development teams. There are a few critical points to consider regarding these processes.

- **The most volatile aspects of any system are the requirements.** And yet most systems contain feature-specific components designed directly against the requirements. This leads to complex systems where changes have rippling effects across many portions of the system. **Coupling a System Design to the current understanding of the requirements is like building a house on sand**. Of course requirements are crucial, but they are also certain to change. Therefore **designing against the requirements guarantees a poor quality design** that becomes increasingly difficult to maintain over time. Conclusion: **never design against the requirements**. Design *for* the requirements, but not directly against them.

- A valid System Design provides **Change Containment** by identifying, isolating, and encapsulating **volatilities.** With adequate **Change Containment**, a valid System Design remains consistent and stable, despite the fact that technologies and requirements constantly change.

- System Design purposely focuses on **design rather than technology choices**. A technology first approach tends to violate the golden hammer cognitive bias, The Law of the Instrument,[1] where wielding a particular tool causes one to view that tool as *the* solution to a problem. Therefore, a technology first approach increases the tendency to circumvent **Change Containment** by coupling a system to a particular technological choice. In contrast, a design-first approach simplifies communication, reduces cognitive burden, and helps prevent vendor and tool lock-in.

- System Design recognizes and embraces the fundamental principle that **features are aspects of integration, not implementation**.
    - A car has no “driving” component. Driving emerges as a feature due to the integration of various components.
    - A kitchen has no “cooking” component. Cooking emerges as a feature due to the integration of various components.

- An initial System Design effort **may not provide a final answer regarding the architecture.** Its assumptions must be validated (or invalidated) early and iteratively. In

[1] https://en.wikipedia.org/wiki/Law_of_the_instrument

the same way, a Project Design must and will flex and change over time as new situations and challenges arise.

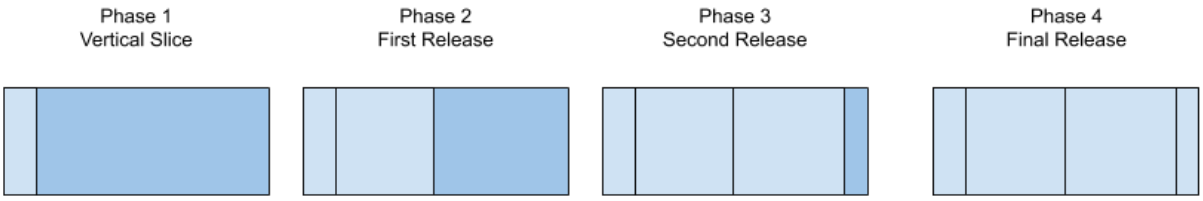
- An effective architect uses System Design and Project Design to provide a **lightweight itinerary** for the architecture and the project plan. A balance is struck between a rigid, unchangeable, approach on one end of the spectrum, and a loose, simplistic, chaotic, approach on the other.[2,3]

- System Design **does not necessitate all new components.** The same processes and techniques may be applied to both greenfield and brownfield systems. Existing tools, components and services may be used to fulfill required capabilities and behaviors so long as **Change Containment** is respected. System Design may also be applied to existing systems in order to evaluate their design validity and complexity.

[2] https://en.wikipedia.org/wiki/Conway%27s_law
[3] https://en.wikipedia.org/wiki/Parkinson%27s_law

- **A System Design can be narrowed in scope to compensate for realities and trade-offs.** A valid System Design accommodates the realities of software product development such as scope expansion and reduction, availability of resources, resource capabilities and experience, budgets, schedules, costs, risks, and customer expectations.

- Each component in a System Design may be **built out in smaller slices**. It is not a requirement to fully implement every possible aspect or facet of each component in a system before they can be integrated and deployed. To be clear, such an approach virtually guarantees the project will not be completed for the lowest possible cost. However there are cases where business priorities and acceptable tradeoffs merit such an approach. Of course, each aspect or facet within a component must also ensure **Change Containment**, properly **encapsulating volatility** at every level of the system.



- Validating the System Design effort with a **Vertical Slice**[4] provides immeasurable advantages to a team. A Vertical Slice involves creating a production quality prototype of the proposed system using production-level infrastructure. This allows the team to test and evaluate non-functional and system quality aspects from the beginning of the development effort. This also enables the **early measurement and monitoring of expected system costs and scalability**. These benefits result in a **massive boost in quality, testability, and maintainability** as well as developer **morale, confidence, and productivity**.

- **System Design is not a silver bullet.** There is no such thing as a free lunch. System Design requires significant creativity, practice and discipline. However, the clear benefits make it worth the effort.

[4] A Vertical Slice involves creating a prototype of a subset of the system components necessary to fulfill a core use case. The Vertical Slice provides the ability to validate critical non-functional aspects of the system before the actual implementation effort begins. This prototype should provide a production quality demonstration of the system infrastructure that can grow and expand, as opposed to a proof-of-concept that would typically be used for reference, but ultimately thrown away.

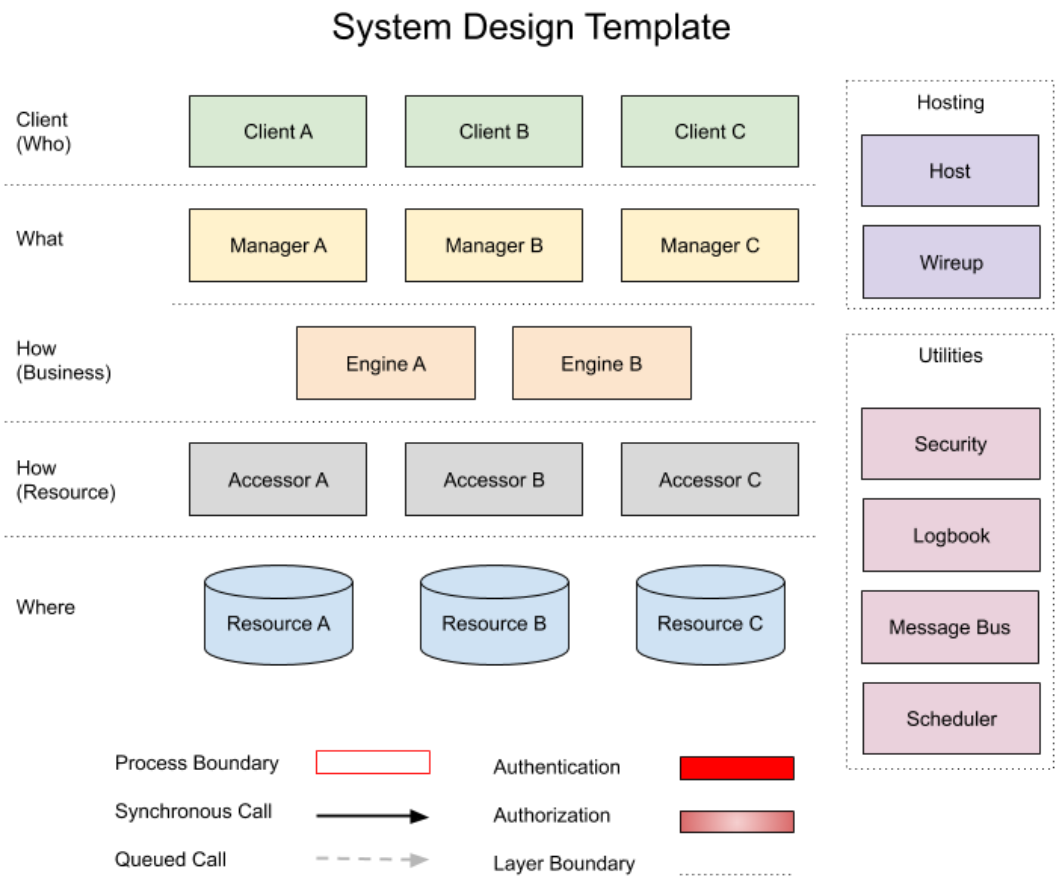
System Design

Be sure to read the included README

System Design Visualizations

System Design Template

The System Design Template provides a legend for how to interpret System Design diagrams.



Green Client components address the “who” aspect of the system, representing client applications such as web, mobile, or third party applications.

Yellow Manager components address the “what” aspect of the system. They are specifically tasked with encapsulating the execution of Use Cases, particularly pertaining, though not limited to, the sequence of operations.

Orange Engine components address the “how” aspect of the system as it pertains to details of the business. They encapsulate the execution of Use Cases in regard to volatility in how actions are performed within the system.

Grey Accessor components address the “how” aspect of the system as it relates to accessing Resources. They define, receive and provide the atomic elements of The System. They also prevent Managers and Engines from becoming dependent on details regarding the nature of Resources.

Blue Resource components represent data storage, external APIs, or sub-systems.

Purple Hosting components appear on the right of the diagram. Host components represent the process and entry point for Manager components and their dependencies. Wireup components encapsulate the volatility of different policies for determining dependency resolution (such as development or testing environments versus production environments).

Pink Utilities also appear on the right of the diagram. Some utilities are meant to be accessed by any System component (a Logbook, for example). Others are specifically meant only for Manager integration (a Message Bus, for example).

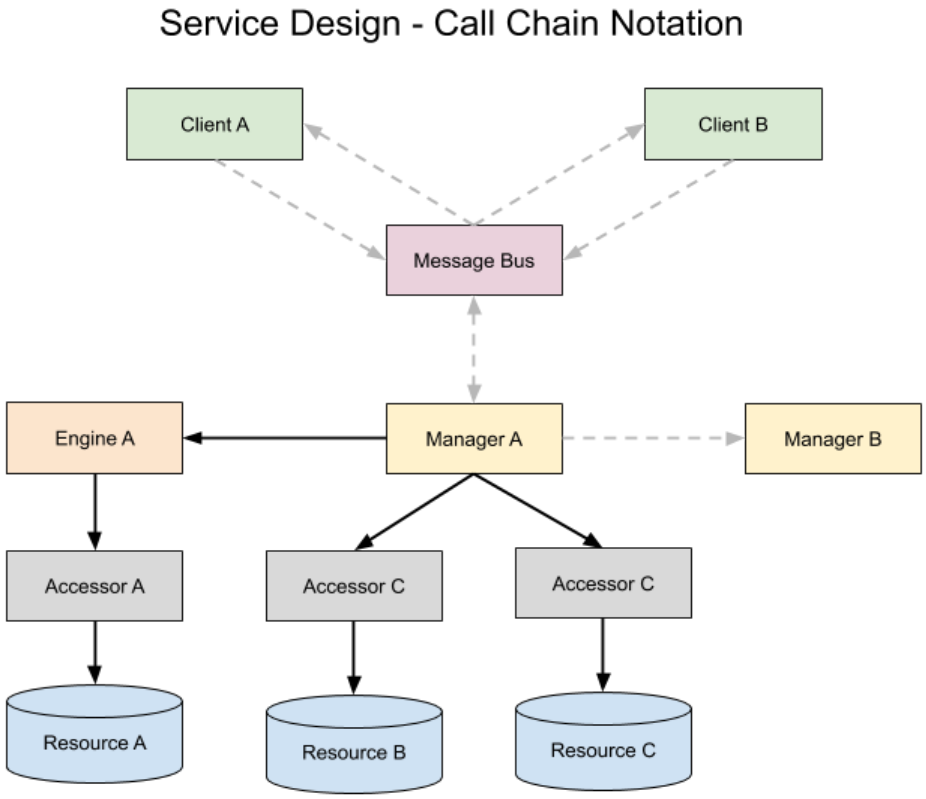
Note also the visual elements representing Authentication, Authorization, Process boundaries, Layer boundaries, and Call arrows.

Keep in mind these additional considerations when creating or interpreting System Design visualizations:

- **No component of The System may call upward to any other component**.
- **No component of The System may call sideways** (the exception being Queued calls between Managers via the Message Bus).
- **Use Case orchestration must happen in The System, and not in Client applications.** This follows from the idea of encapsulating volatilities. Client orchestration has the same issue as designing feature-specific components - changes ripple across the system rather than remaining isolated within components.
- **A Client must be able to complete a Use Case by interacting with one-and-only-one Manager** (the exception is if the orchestrating manager provides an address/endpoint that the client can use to continue the Use Case flow).

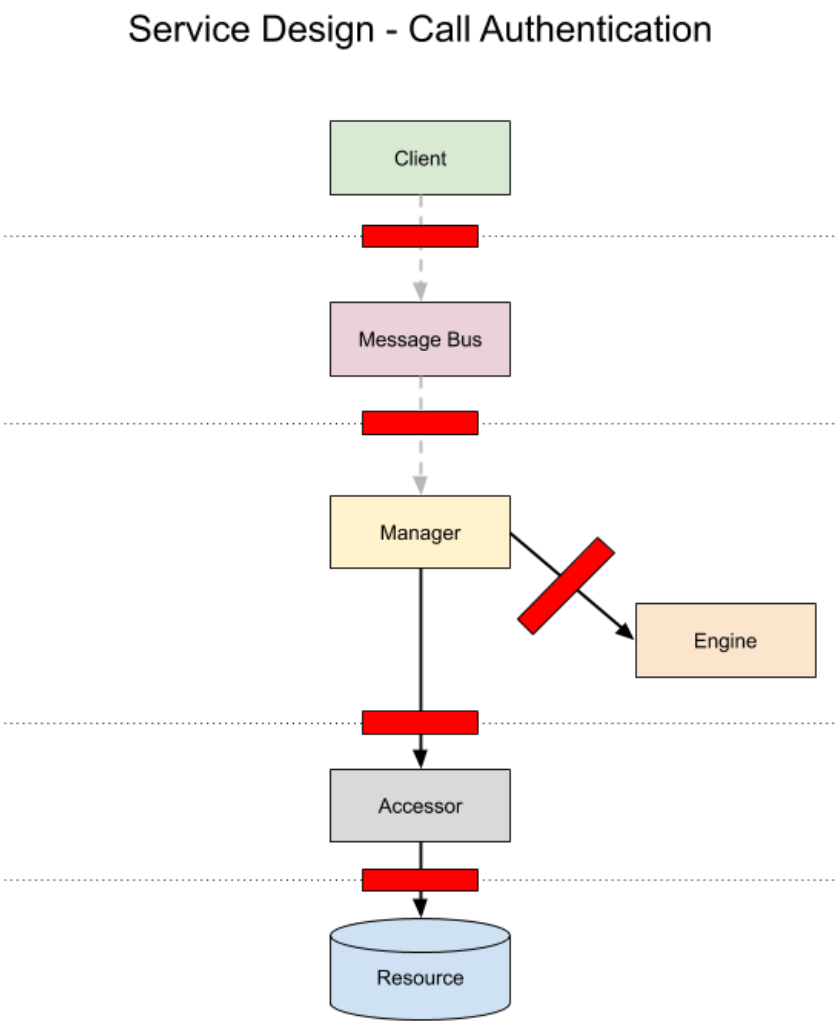
Call Chain Notation

The Call Chain view represents the nature of a prototypical interaction between client applications and The System. Note in particular the use of black solid arrows to represent synchronous calls, and grey dashed arrows to represent asynchronous or queued calls.



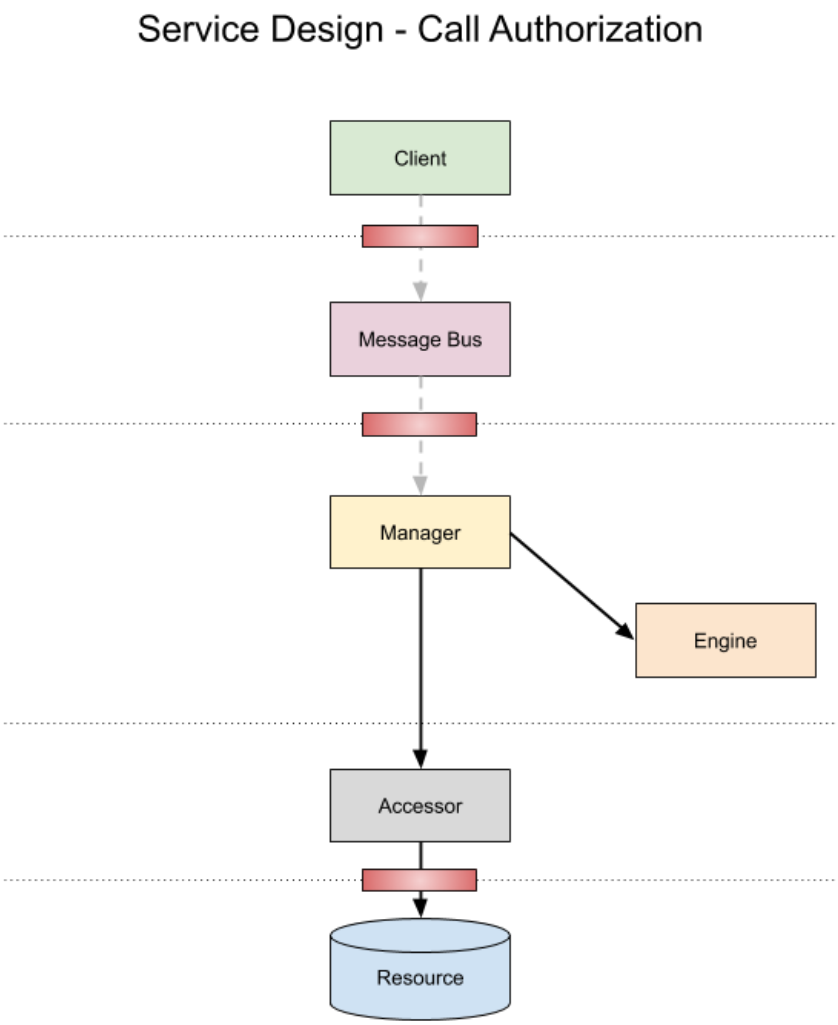
Call Authentication Notation

The Call Authentication view demonstrates that every logical layer crossing in The System is authenticated, and every cross process call is authenticated. In-process calls are also authenticated for purposes of message protection.



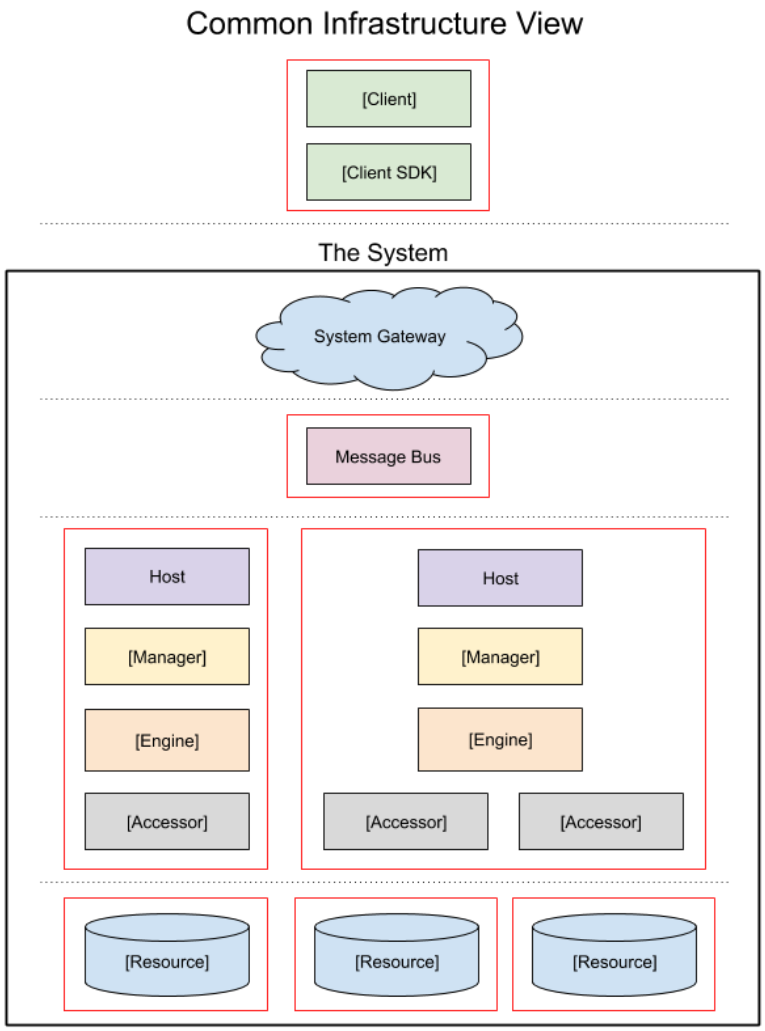
Call Authorization Notation

The Call Authorization view demonstrates that every logical layer crossing in The System is authorized, and every cross process call is authorized. In-process calls, however, do not require authorization since they share identity and communicate only using private pipes or sockets.



Common Infrastructure View

The Common Infrastructure View provides an abstract, generalized view of a typical System infrastructure. Though not absolutely required, the common infrastructure recommends that Clients interact with The System through Client SDKs. All Client SDK requests to The System are dispatched via a Message Bus by way of a System Gateway. Red boundaries represent prototypical process boundaries.



Presenting the infrastructure in this generalized fashion allows System Design diagrams to avoid the need to repeat the appearance of Client SDKs, the System Gateway, the Message Bus, or Host processes. Their presence may be implied for any interaction between a Client and a Manager, by way of The System, via the Message Bus.

Project Design

Be sure to read the included README

Some general notes on the included spreadsheets:

- As a rule, yellow cells are meant for you to modify
- As a rule, grey cells are not meant to be modified
- As with all spreadsheets, copy/paste/insert/delete can cause strange behaviors. Start slow in the beginning until you get a sense of how to avoid breaking things terribly and then not noticing until it's too late to undo. Murphy's law - can happen, or will happen?

Project Design and Tracking

The Right Design Project Design Template is the architect's starting point for Project Design. The **Activities** sheet contains a **Project Start** date, and cells for entering project activities. **Dependencies** and **Resources** can also be managed in their corresponding sheets.

Activity Network Validation

A great way to validate the project activity network is with a mermaid-js dependency diagram (it's really a flowchart, but it gets the job done).

The content from the **Mermaid** sheet goes straight into the Mermaid Live Editor and voila! Excessive integration pressure, dangling activities, and incorrect or duplicated dependencies can be visually identified.

Of course the default colors aren't extremely helpful. Copying the **Class Defs** column of the **Config** sheet into the Mermaid Live Editor provides a much clearer, prettier output.

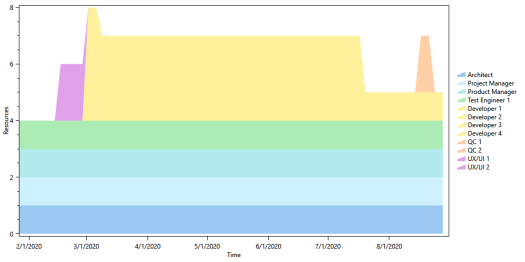
Ahh. That's nice. :)

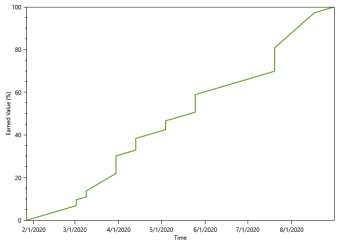
Project Planning

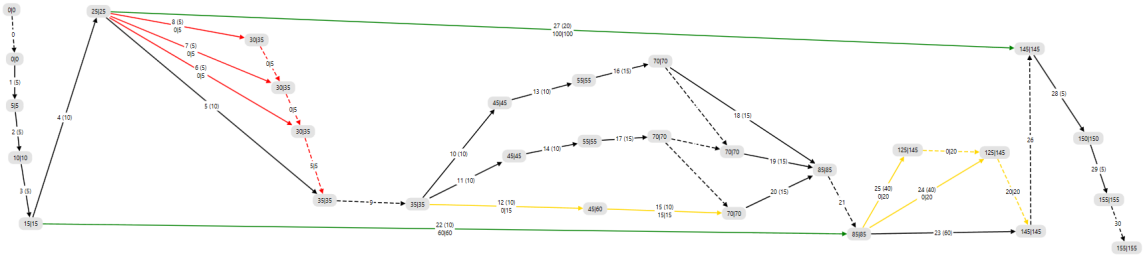
The **OutputJSON** sheet creates a JSON format representing your project design. Copy that text and paste it into a text file with a .json extension.

Using the Project Plan Application, choosing File→Import... allows the previously saved .json file to be imported. The Project Plan Application immediately goes to work calculating risk, duration, costs, floats, and expected delivery dates.

The Project Plan Application also generates charts for staffing distribution, expected earned value, and a project network arrow diagram.







The architect can now explore various options for planning the project.

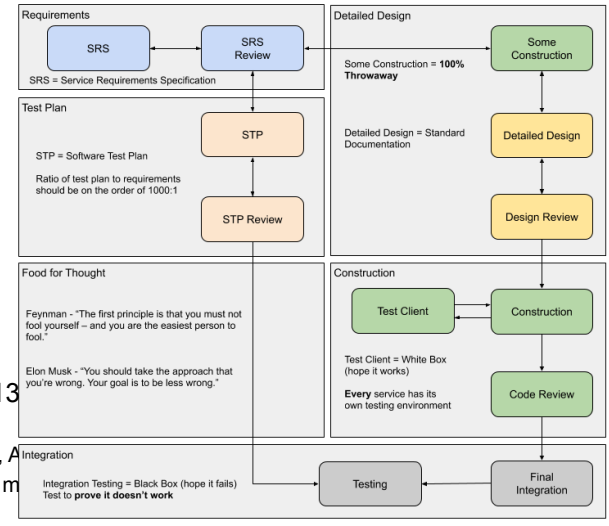
Project Tracking

The Right Design Project Tracking Template enables detailed project plan tracking. Tracking the project allows architects and project managers to visualize project progress (or lack thereof) and make informed decisions about current and future project resourcing options.

After the Software Development Plan Review (an SDP Review happened, right?) the architect opens the .zpp file for the approved project using the Project Plan Application. Choosing File→Export CSV… creates a comma separated values file containing the vital information about the approved project.

The contents of the csv file should entirely replace the contents of the **Raw Activities** sheet of the Right Design Project Tracking Template.

The **README** sheet enables modification of the project **Tracking Resolution**. Typical



values will be 7 days (1 week), 14 days (2 weeks), or 28 days (4 weeks). It is recommended to use a *smaller resolution for shorter projects*, and a *larger resolution for longer projects*. The **Tracking Resolution** defines the time interval length for which project task completion will be tracked. The **Finished By** dates in the **Project Tracking** sheet are calculated based on the project **Start Date** and the specified **Tracking Resolution**.

Each activity of greater than zero days in **Duration** should be comprised of one or more tasks necessary for completing the activity. Enter the number of tasks for each activity in the **Task Count** column of the **Project Tracking** sheet. For example, a typical service activity should follow the standard Service Development Lifecycle which involves 5 tasks - 1) Requirements, 2) Detailed Design, 3) Test Plan, 4) Construction, and 5) Integration.

Each completed task will be **Finished On or Before** one of the dates specified in **Row 1** of the Project Tracking sheet. Each date heading contains two columns beneath it. Column **P** is for entering Progress as the number of completed tasks. Column **E** is for entering Effort Days.

The number of tasks completed for an activity by the **Finished On or Before** date are entered in the corresponding cell. In the example below, the Requirements activity is comprised of a single task with an estimated duration of 5 days. Cell O4 contains a “1” for the Progress of one completed task. Call P4 contains a “5” for the number of Effort Days it took to complete the Activity.

Coming Soon...

More information is forthcoming regarding usage, and how to integrate with the Project Plan Application.

Additional Resources

Templates

Visit https://jeffdoolittle.com/right_design_templates for access to System Design templates and other useful resources you can use for your own System and Project Design efforts.

Required Reading

Righting Software
Juval Löwy
https://www.amazon.com/Righting-Software-Juval-Löwy/dp/0136524036

On the Criteria To Be Used in Decomposing Systems into Modules
David Parnas
https://www.win.tue.nl/~wstomv/edu/2ip30/references/criteria_for_modularization.pdf

Escaping Appland
Michael “Monty” Montgomery
https://itabok.iasaglobal.org/from-the-field-escaping-appland/

Law of the Instrument
“I suppose it is tempting, if the only tool you have is a hammer, to treat everything as if it were a nail.”
https://en.wikipedia.org/wiki/Law_of_the_instrument

Conway’s Law
“Organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations.”
https://en.wikipedia.org/wiki/Conway%27s_law

Parkinson’s Law
“Work expands so as to fill the time available for its completion.“
https://en.wikipedia.org/wiki/Parkinson%27s_law