COMPLEX REQUIREMENTS & SOLUTIONS CHECKLIST

# Introduction

**Guidance for Completion of Design Analysis Document**

General Guidance and Purpose:

* This checklist is broken down into multiple sections:
  + Part A – Requirements Breakdown (“What We Build”)
  + Part B – Solutions Breakdown (“How We Build It”)
* Work through the process of analysis by writing things down
* Time box your effort to an hour or two
* Use whatever format works best for you (pictures, words, flowcharts, etc)
* Some sections might not apply. Feel free to skip them.
* Final section should be a set of implementation tasks with estimates

# Part A –Requirements Breakdown (“What We Build”)

**What is the Problem?**

*Define the problem without any reference to possible solutions. Should be in user language from the user’s point of view.*

**Definitions:**

*What terms need to be defined to get everyone on the same page.*

**How are people currently solving the problem?**

*What hacks are people using to get around the problem today.*

**What is my understanding of the requirements? What is the primary goal or objective?**

*Use this section to articulate, in your words, what the intent is of this feature/requirement and how it is intended to work. Include what the desired impact/capability will be for the intended user or beneficiary of this feature. Remember, the user should drive the system’s functionality.*

**What are the secondary goals/objectives?**

*List any secondary objectives or goals that are separate from the primary objective.*

**What are the unknowns? (i.e. what information is not currently available?)**

*In addition to uncovering requirements that you can make assumptions about, you might also uncover something that is completely unknown that could have an impact on development once you get started. For example, maybe there is an API that you need to use but the definition of that API is not published yet. Or, maybe there is an algorithm that needs to be developed that it is not known whether said algorithm is even feasible.*

**What parties are impacted by the work?**

*What individuals, groups, users, roles are involved in the work.*

**What are the known design constraints?**

*What are the limitations that have been imposed on the implementation of this feature that reduce/impact the design decision choices you can make? Constraints can come from a variety of sources: cost, time, hosting, user requirements, performance, etc. In general, design constraints are a valuable tool to reduce the scope of possible solutions to a problem so identifying these constraints early can help guide us to the best design decisions.*

**What are the acceptance criteria for the solution (including non-functional)?**

*Enumerate the list of criteria by which the success or failure of your implementation should be judged. Be sure to include non-functional criteria such as performance, quality, reliability, resource utilization, etc.*

**What assumptions am I making about these requirements and the system?**

*Often times when you are going through this process of analyzing and writing down how you are going to build something, you uncover something you had not considered that must be addressed even though it is not spelled out in the requirements. For example, if the requirement is “charge the credit card as part of a product fulfillment process” you might ask yourself, “What happens if the attempt to charge the card fails for some reason?” Another assumption might be that you will be able to test this credit card workflow without using a real credit card. This is where you can start listing your assumptions about how you should be handling some of these undocumented requirements so that other stakeholders can review and validate these assumptions.*

**What are the priorities I am deliberately making?**

*If there are many options to go after, what are we prioritizing as the best first use cases and why? These items will be listed in order of their priority from left to right.*

**What are the tradeoffs I am deliberately making?**

*We are always making tradeoffs. Most of the time it feels like these are implicit and not explicit. The goal of this section is to try to get the engineer to identify tradeoffs they may be making in their approach. Consistency, performance, maintainability, testability, etc?*

**What areas/features are most likely to change over time?**

*As you are analyzing these requirements it will be important for you to assess what areas of the software are likely to change as a result of testing difficult areas of the code, through user feedback, new feature development down the road, etc. This information will inform how your design should be encapsulating these volatilities.*

**What areas/features/components are most likely not to change?**

*These are typically components that are so rooted in the concept, problem, or domain that they will not change much over time. This can involve things that have been working the exact same and working well for such a long time that they probably don’t need to be adjusted.*

**What are anticipated/possible failure scenarios and how should they be handled?**

*The purpose of this section is to get you thinking about how the system might behave once you get off the “happy path” of execution.*

*What happens if an external dependency (service, API, etc) fails or does not respond as expected.*

*What if inputs to the logic are outside your expectations?*

*What if the system comes under an unexpected load?*

*Where and how should exceptions be handled?*

*If a multi-step process fails at some mid-point, what should be the steps to recovery, completion, and/or return to a consistent state?*

*Do I need to use some sort of persisted multi-state flag to determine progress?*

*Should I put in some strategic logging to give details when certain failures occur?*

Do I have any potential race conditions?

**Are there any special considerations related to security?**

*Do I need to authenticate and/or authorize the user accessing the business logic?*

*Is there any special treatment of sensitive data that should be taken into account?*

*Am I potentially exposing sensitive data in exceptions logs, etc?*

**What are the existing areas impacted?**

*This section is intended to demonstrate identifying the modules/classes/services that are impacted by this feature along with the types of changes that will be made to these components. This section should also be used to identify service and data contract modifications that will be required.*

**How will the design/architecture need to change?**

*Are we adding any new manages/engines/accessors/utilities?*

*Are we creating any new interfaces or data contracts?*

**How can I encapsulate current and future change?**

*The purpose of this section is to get you to analyze the current feature/requirement in terms of the likelihood that it will change in the future. The idea is to help you think through what the best approach is for encapsulating future change.*

**How should I test and validate the system?**

*What will be your strategy for deciding how to validate the system performs as expected? Aside from writing unit tests, are there specific integration tests that you will want to develop/modify to verify this requirement? What are some edge cases you will want to make sure you are testing?*

**What risks am I aware of?**

*Do you feel there is uncertainty in the outcome of this effort?*

*Is there any risk related to the estimates that have been provided?*

*Is there a level of requirements ambiguity that could result in significant re-work?*

*Are you missing any information required to fully understand the requirements (e.g. UI/UX designs/workflows, algorithm specifications, etc?)*

*Are we using new technologies that we are not familiar with?*

*Are you concerned about the stability/robustness of technologies or third part dependencies?*

*Do any of the assumptions above represent significant risk if they are not valid?*

**What concerns do I have?**

*Is there anything else that you can think of that you would want to surface in this document?*

**What are the specific steps to implement and what is their level of effort?**

*This will become the sequential plan for implementing this feature/requirement and will include the “final” estimate of the level of effort to complete each task. In addition to specific feature development tasks, other things to consider here would be:*

*New/modified data contracts*

*New/modified service contracts*

*New/modified unit and integration tests*

*Manual testing efforts*

**Specific Functional Requirements**

*Are all inputs specified including their source, accuracy, range of values and frequency?*

*Are all outputs specified including their destination, accuracy, range of values, format, and frequency?*

*Are all the external hardware and software interfaces specified?*

*Are all the external communication interfaces specified, including handshaking, error checking, and communication protocols.*

*Are all the tasks the user wants to perform specified.*

*Is the data used in each task and the data resulting from each task specified?*

**Specific Nonfunctional (Quality) Requirements**

*IS the expected response time, from the user’s point of view, specified?*

*Are other timing considerations such as processing time, data transfer rate, and system throughput specified?*

*Is the level of security specified?*

*Is the reliability specified, including consequences of software failure, vital information that needs to be protected from failure, and strategy for error detection and recovery?*

*Are the minimum machine memory and free disk space specified?*

*Is the maintainability of the system specified, including its ability to adapt to changes in specific functionality, changes in the operating environments, and changes in its interfaces with other software?*

**Requirements Quality**

*Do requirements conflict?*

*Do the requirements avoid specifying the design?*

*What areas are we lacking detail?*

*Are the requirements clear enough to be handed over to a third party group to be understood.*

*Is each requirement testable?*

*Is each item relevant to the problem?*

**Requirements Completeness**

*Where information isn’t available before development begins, are the areas of incompleteness specified?*

*Are the requirements complete in the sense that if the product satisfies every requirement, it will be acceptable?*

*Are you comfortable with the requirements?*

# Part B – Solutions Breakdown (“How We Build It”)

**Brainstorm Possible Solutions - Alternatives**

*Identify some solutions. Focus on quantity. No criticism. Encourage wild ideas. Combine and improve ideas. Sketch where all can see.*

**Design Trees**

*Modeling decision criteria.*

**Test Possible Solutions**

*Try your possible solutions with a few basic tests. What is the minimal definition you can give to see if this will work as expected. Often, you identify the hardest part of the solution and make a simplified proof of concept.*

**Categories**

*Identify categories of solution. Many solutions are qualitatively the same. Try to pick out what makes one solution different from another. Small details can be decided and changed later.*

**Category Comparison**

*Examine the qualitative difference of the categories. Outline some pros and cons. Identify what information would let you make a choice and devise the simplest test you can to get the info. Pick a category. Repeat this with sub categories until you have one high level solution. If you run into conflicts go backup the category tree to the first level that isn’t subject to that failure and pick a new path.*

**More Information/Research**

*What information are we missing to help make the decision?*

**Proposed Solution**

*What does your best solution look like weighing in all the costs and benefits?*

**Test Proposed Solution**

*Try your proposed solution with a few basic tests. What is the minimal definition you can give to see if this will work as expected. Often, you identify the hardest part of the solution and make a simplified proof of concept. \*\*List sample size on tests (if applicable).*

**Contingency Planning**

*Are there any elements of the plan that would force a new path if they failed? Prove them if you can. Identify code components as an outline or a sketch on paper. Research code libraries/ tooling.*

**Feedback**

*Get feedback from relevant parties involved.*

**Reconsider Problem**

*If at any point it feels like there is no attainable solution or that the solutions will not get us to our company goals, reconsider the problem. If you can’t easily identify criterion or solutions, you may be solving the wrong problem or your problem may not have enough definition. Half the battle (or more) can be solving the right problem.*

**Major Classes**

*Responsibilities of each class and how they interact with other classes. How are classes organized into subsystems. Aim for 80/20 rule. Specify the 20% of classes that make up 80% of system behavior.*

**Data Design**

*Major files and table design to be used. How is data accessed.*

**Business Rules**

*Identify specific business rules and impact on system design.*

**User Interface Design**

*Major elements of web pages formats. Should be modularized so that new interfaces can be substituted without affecting the business rules.*

**Resource Management**

*Plan for managing scares resources like database connections, threads, handles, etc.*

**Security**

*Approach to design-level and code-level security.*

**Performance**

*Performance goals including resource use, what has priority, including speed v. memory v. cost. Should explain why certain performance goals are achievable.*

**Scalability**

*How the system will address growth in number of users, servers, network nodes, database records, transactions, etc.*

**Interoperability**

*How the system will share data or resources with other software or hardware.*

**Internationalization/Localization**

*Preparing a program to support multiple locales and languages or a local locale and language. Inlcuding Prompts, status displays, help messages, error messages, character format and string format.*

**Input/Output**

*How input and output processing and errors are handled.*

**Error Processing**

Including:

*Is error processing corrective (recovers from errors) or merely detective (processes as if nothing happened or quit)?*

*Is error detection active (anticipate errors, checking input) or passive (respond to errors only when it can’t avoid)?*

*How does the program propagate errors? Discard the data, enter error processing state, wait and notify the user?*

*What are the conventions for handling errors? What is single consistent strategy?*

*How will exceptions be handled? When the code can throw exceptions, where they will be caught, how they will be logged, how they will be documented.*

*At what level are errors handled? Are they at point of detection or passed up call chain?*

*Level of responsibility of each class for validating it’s input data? Can be done at a class level or system level.*

*Use environments built-in exception handling or build your own.*

**Fault Tolerance**

*Techniques to detect errors, recover from them, and contain their bad effects (backup and try again, backup values, algorithm, replace with default value):*

**Buy v. Build Decisions**

*What third party off the shelf components should be used.*

**Reuse Decision**

*What previously built technology can be reused.*

**Ambient Context**

*How are your passing in common context that is needed throughout the application.*

**Naming**

*How are you naming systems, classes, contracts, files, methods, tests, etc.*

**Testing**

*What tests are needed for the solution?*

**Design Patterns/Architectural Patterns**

*What are some of the consistent design and architectural patterns for the solution?*

**Modeling**

*Call Chain*

*Static Diagram*

**General Software Engineering Principles**

*Does anything violate SOLID*

*Cohesion*

*De-Coupling*

*Information Hiding*

**Coding Language**

*Team’s experience and familiarity*

*Language popularity in target industry*

*Front End*

*Back End*

*Database*

*High Level v. Low Level Language*

**Major Construction Practices**

*Conventions for names, comments, and layouts.*

*Conventions for Common Routine Operations*

*Convention for dependency inversion (service locator, dependency injection)*

**Teamwork Practices**

*Integration procedure – how will team members check code into the master source?*

*What environments will be setup (local, dev, QA, prod)?*

*Will programmers work in pairs?*

**Quality Assurance**

*Will programmers write test cases for their code prior to or after writing code?*

*Types of tests (unit, function, integration, user, end-to end, alpha, beta, operational, regulation, security, requirements)*

*Code review*

**Tools**

*How will you control and integrate versions?*

*Development Tools*

*Frameworks*

*Nonstandard language features*