checkcheckzz / **system-design-interview**

System design interview for IT companies

#interview #interview-questions #interview-preparation #design-systems #system

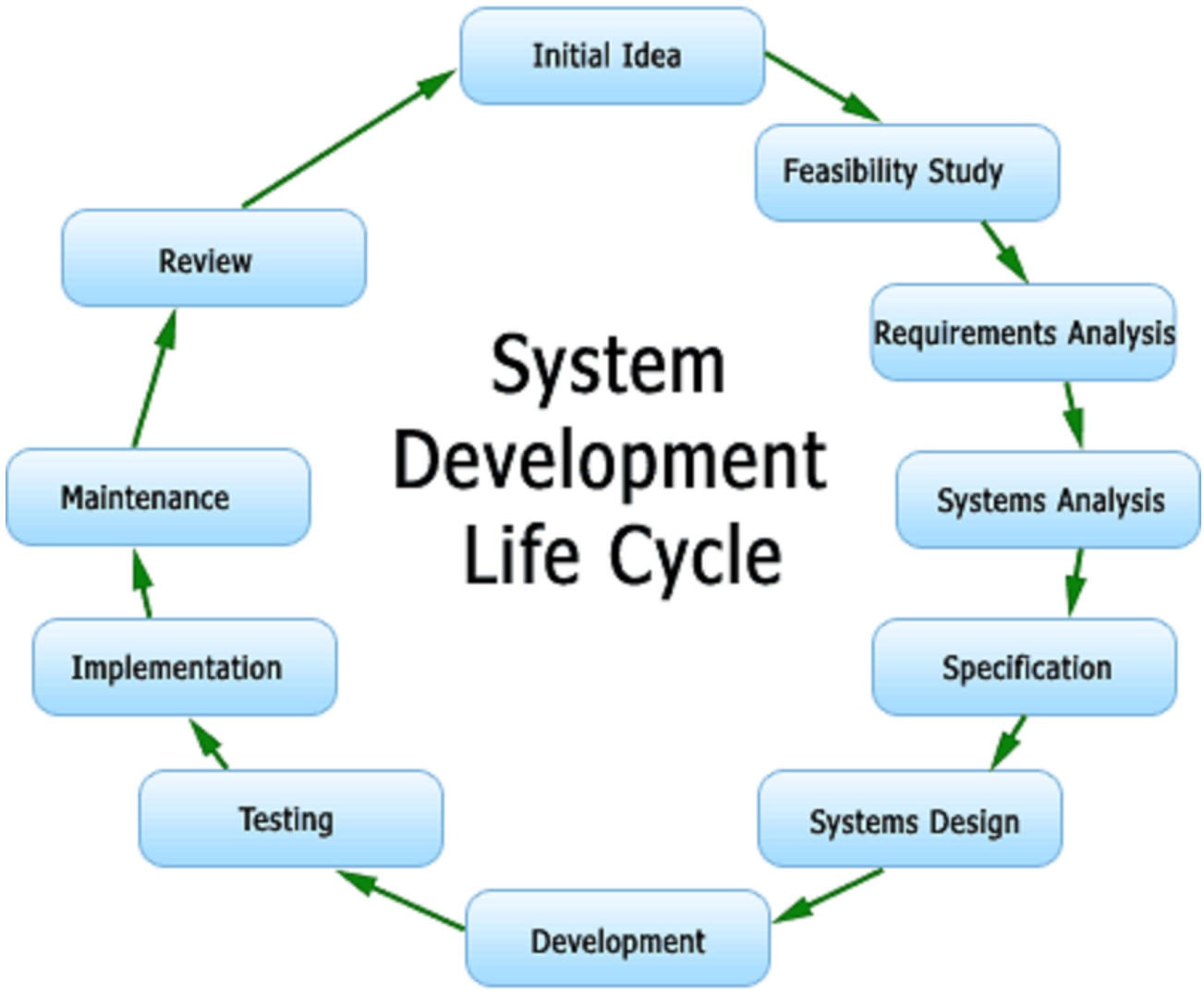
65 commits | 1 branch | 0 releases | 15 contributors

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

checkcheckzz Merge pull request #30 from praveenHp/master ··· Latest commit 9e9e754 on Feb 3

imgs	add logo	4 years ago
README.md	Added Link to Scalyr Dev blog	4 months ago

README.md



> How to prepare system design questions for an IT company

System design is a very broad topic. Even a software engineer with many years of working experience at a top IT company may not be an expert on system design. If you want to become an expert, you need to read many books, articles, and solve real large scale system design problems.

This repository only teaches you how to handle the system design interview with a systematic approach in a short time. You can dive into each topic if you have time. Of course, welcome to add your thoughts!

Table of Contents

[⬆] System Design Interview Tips:

Clarify the constraints and identify the user cases

Spend a few minutes questioning the interviewer and agreeing on the scope of the system. Remember to make sure you know all the requirements the interviewer didn't tell you about in the beginning.

User cases indicate the main functions of the system, and constraints list the scale of the system such as requests per second, requests types, data written per second, data read per second.

High-level architecture design

Sketch the important components and the connections between them, but don't go into some details. Usually, a scalable system includes webserver (load balancer), service (service partition), database (primary/secondary database cluster plug cache).

Component design

For each component, you need to write the specific APIs for each component. You may need to finish the detailed OOD design for a particular function. You may also need to design the database schema for the database.

[⬆] Basic Knowledge about System Design:

Here are some articles about system design related topics.

- How to Rock a Systems Design Interview
- System Interview
- Scalability for Dummies
- Scalable Web Architecture and Distributed Systems
- Numbers Everyone Should Know
- Fallacies of distributed systems
- Scalable System Design Patterns
- Introduction to Architecting Systems for Scale
- Transactions Across Datacenters
- A Plain English Introduction to CAP Theorem
- The CAP FAQ
- Paxos Made Simple
- Consistent Hashing
- NOSQL Patterns
- Scalability, Availability & Stability Patterns

Of course, if you want to dive into system related topics, here is a good collection of reading list about services-engineering, and a good collection of material about distributed systems.

[⬆] Company Engineering Blogs:

If you are going to have an onsite with a company, you should read their engineering blog.

- High Scalability
- The GitHub Blog
- Engineering at Quora
- Yelp Engineering Blog
- Twitter Engineering
- Facebook Engineering
- Yammer Engineering
- Etsy Code as Craft
- Foursquare Engineering Blog
- Airbnb Engineering
- WebEngage Engineering Blog
- LinkedIn Engineering
- The Netflix Tech Blog
- BankSimple Simple Blog
- Square The Corner
- SoundCloud Backstage Blog
- Flickr Code
- Instagram Engineering
- Dropbox Tech Blog
- Cloudera Developer Blog
- Bandcamp Tech
- Oyster Tech Blog
- THE REDDIT BLOG
- Groupon Engineering Blog
- Songkick Technology Blog
- Google Research Blog
- Pinterest Engineering Blog
- Twilio Engineering Blog
- Bitly Engineering Blog
- Uber Engineering Blog
- Godaddy Engineering
- Splunk Blog
- Coursera Engineering Blog
- PayPal Engineering Blog
- Nextdoor Engineering Blog
- Booking.com Development Blog
- Scalyr Engineering Blog

[⬆] Products and Systems:

The following papers/articles/slides can help you to understand the general design idea of different real products and systems.

- MapReduce: Simplied Data Processing on Large Clusters
- Bigtable: A Distributed Storage System for Structured Data
- The Google File System

- The Chubby lock service for loosely-coupled distributed systems
- Dynamo: Amazon's Highly Available Key-value Store
- Introduction to Memcached
- Cassandra Introduction Features
- Introduction to HBase
- Introduction to MongoDB
- Introduction to Redis
- Storm
- Introduction to Zookeeper
- Kafka
- YouTube Architecture
- Scaling Pinterest
- Google Architecture
- Scaling Twitter
- The WhatsApp Architecture
- Flickr Architecture
- Amazon Architecture
- Stack Overflow Architecture
- Pinterest Architecture
- Tumblr Architecture
- Instagram Architecture
- TripAdvisor Architecture
- Scaling Mailbox
- Salesforce Architecture
- ESPN Architecture
- Uber Architecture
- DropBox Design
- Splunk Architecture

[⬆] Hot Questions and Reference:

There are some good references for each question. The references here are slides and articles.

Design a CDN network
Reference:

- Globally Distributed Content Delivery

Design a Google document system
Reference:

- google-mobwrite
- Differential Synchronization

Design a random ID generation system
Reference:

- Announcing Snowflake
- snowflake

Design a key-value database
Reference:

- Introduction to Redis

Design the Facebook news feed function
Reference:

- What are best practices for building something like a News Feed?
- What are the scaling issues to keep in mind while developing a social network feed?
- Activity Feeds Architecture

Design the Facebook timeline function
Reference:

- Building Timeline
- Facebook Timeline

Design a function to return the top k requests during past time interval
Reference:

- Efficient Computation of Frequent and Top-k Elements in Data Streams
- An Optimal Strategy for Monitoring Top-k Queries in Streaming Windows

Design an online multiplayer card game
Reference:

- How to Create an Asynchronous Multiplayer Game
- How to Create an Asynchronous Multiplayer Game Part 2: Saving the Game State to Online Database
- How to Create an Asynchronous Multiplayer Game Part 3: Loading Games from the Database
- How to Create an Asynchronous Multiplayer Game Part 4: Matchmaking
- Real Time Multiplayer in HTML5

Design a graph search function
Reference:

- Building out the infrastructure for Graph Search
- Indexing and ranking in Graph Search
- The natural language interface of Graph Search and Erlang at Facebook

Design a picture sharing system
Reference:

- Flickr Architecture
- Instagram Architecture

Design a search engine
Reference:

- How would you implement Google Search?
- Implementing Search Engines

Design a recommendation system
Reference:

- Hulu's Recommendation System
- Recommender Systems

Design a tinyurl system
Reference:

- System Design for Big Data-tinyurl
- URL Shortener API

Design a garbage collection system
Reference:

- Baby's First Garbage Collector

Design a scalable web crawling system
Reference:

- How can I build a web crawler from scratch?

Design the Facebook chat function
Reference:

- Erlang at Facebook
- Facebook Chat

Design a trending topic system
Reference:

- Implementing Real-Time Trending Topics With a Distributed Rolling Count Algorithm in Storm
- Early detection of Twitter trends explained

Design a cache system
Reference:

- Introduction to Memcached

[⬆] Good Books:

- Big Data: Principles and best practices of scalable realtime data systems
- Real-Time Analytics: Techniques to Analyze and Visualize Streaming Data
- Building Microservices: Designing Fine-Grained Systems
- Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems

[⬆] Object Oriented Design:

Tips for OOD Interview

Clarify the scenario, write out user cases

Use case is a description of sequences of events that, taken together, lead to a system doing something useful. Who is going to use it and how they are going to use it. The system may be very simple or very complicated.

Special system requirements such as multi-threading, read or write oriented.

Define objects

Map identity to class: one scenario for one class, each core object in this scenario for one class.

Consider the relationships among classes: certain class must have unique instance, one object has many other objects (composition), one object is another object (inheritance).

Identify attributes for each class: change noun to variable and action to methods.

Use design patterns such that it can be reused in multiple applications.

Useful Websites

- 101 Design Patterns & Tips for Developers