Christian Vielma

Interview Study Guide – Software Development and Architecture

V1.1

Disclaimer

I (Christian Vielma) have personally created this guide for personal use, to be used as a reference for technical interviews, work, and study. Since I believe this document might provide value to others, I have decided to make it public.

Most of the content on this document have been added from different sources, from freely available resources on the Internet, to books, with additional added content from me. I made my best effort to provide full credit to the original sources, and in no way I'm trying to take advantage of improper quotations. If you believe I have made an invalid reference to a resource, please let me know and I'll fix it.

I provide this guide as-is, with no additional guarantees. I'm also releasing it under CC-Atribution-ShareAlike License, so you are free to extend it and update it with proper attribution.

Enjoy!

Table of Contents

Software Development Best Practices

(RUP) https://en.wikipedia.org/wiki/IBM_Rational_Unified_Process#Six_Best_Practices

- Develop Iteratively
- Manage Requirements
- Use Components
- Model Visually
- Verify Quality
- Control changes

Object Oriented Analysis and Design

References:

- McLaughlin, Pollice, West. 2006. Head's First Object Oriented Design and Analysis.
- Mongan, Giguere, Kindler. 2012. Programming Interviews Exposed: Secrets to Landing your Next Job.

Concepts

- **Class:** is an abstract definition of something that has attributes (sometimes called properties or states) and actions (capabilities or methods).
- **Object:** is a specific instance of a class that has its own state separate from any other object instance.
- **Inheritance:** allows a class to be defined as a modified or more specialized version of another class.
- **Polymorphism:** is the capability to provide multiple implementations of an action and to select the correct implementation based on the surrounding context.

Virtual Method

In OOP, child classes can override (redefine) methods defined by ancestor classes. If the method is virtual, the method definition to invoke is determined at run time based on the actual type (class) of the object on which it is invoked. Nonstatic Java methods are always virtual, but in C# and C++methods are only virtual when declared with the virtual keyword — nonvirtual methods are the default. If the method is not virtual, then the method definition invoked is determined at compile time based on the type of the reference (or pointer).

Multiple Inheritance

Problems:

- Ambiguity (as Class D that inherits from B, C that inherits from A).
- How is constructed the object.

Preventing multiple inheritance avoid sharing functionality among classes.

Requirements

- Good requirements ensure your system works like your customers expect.
- Make sure your requirements cover all the steps in the use cases for your system.
- Use your use cases to find out about things your customers forgot to tell you.
- Your use cases will reveal any incomplete or missing requirements that you might have to add to your system.
- Your requirements will always change (and grow) over time.

Analysis and Design

- Well-designed software is easy to change and extend.
- Use basic OO principles like encapsulation and inheritance to make your software more flexible.
- If a design isn't flexible, then CHANGE IT! Never settle on bad design, even if it's your bad design that has to change.
- Make sure each of your classes is cohesive: each of your classes should focus on doing ONE THING really well.
- Always strive for higher cohesion as you move through your software's design cycle.

OO Basics

- Abstraction.
- Encapsulation.
- Polymorphism.
- Inheritance.

SOLID Principles

From: https://en.wikipedia.org/wiki/SOLID_%28object-oriented_design%29

- **S: Single responsibility principle** - a class should have only a single responsibility (i.e. only one potential change in the software's specification should be able to affect the specification of the class)
- **O: Open/closed principle -** "software entities … should be open for extension, but closed for modification."
- **L: Liskov substitution principle -** "objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program." See also design by contract.
- **I: Interface segregation principle -** "many client-specific interfaces are better than one general-purpose interface."
- **D: Dependency inversion principle -** one should "Depend upon Abstractions. Do not depend upon concretions."

Solving Big Problems

- Listen to the customer, and figure out what they want you to build.
- Put together a feature list, in language the customer understands.
- Make sure your features are what the customer actually wants.
- Create blueprints of the system using use case diagrams (and use cases).
- Break the big system up into lots of smaller sections.
- Apply design patterns to the smaller sections of the system.
- Use basic OOA&D principles to design and code each smaller section.

Development Approaches

- **Use case driven development:** takes a single use case in your system, and focuses on completing the code to implement that entire use case, including all of its scenarios, before moving on to anything else in the application.
- **Feature driven development:** focuses on a single feature, and codes all the behavior of that feature, before moving on to anything else in the application.
- **Test driven development:** writes test scenarios for a piece of functionality before writing the code for that functionality. Then you write software to pass all the tests.

Good software dev usually incorporates **all** of these development models at different stages of the development cycle.

Design Patterns

From: http://www.oodesign.com/

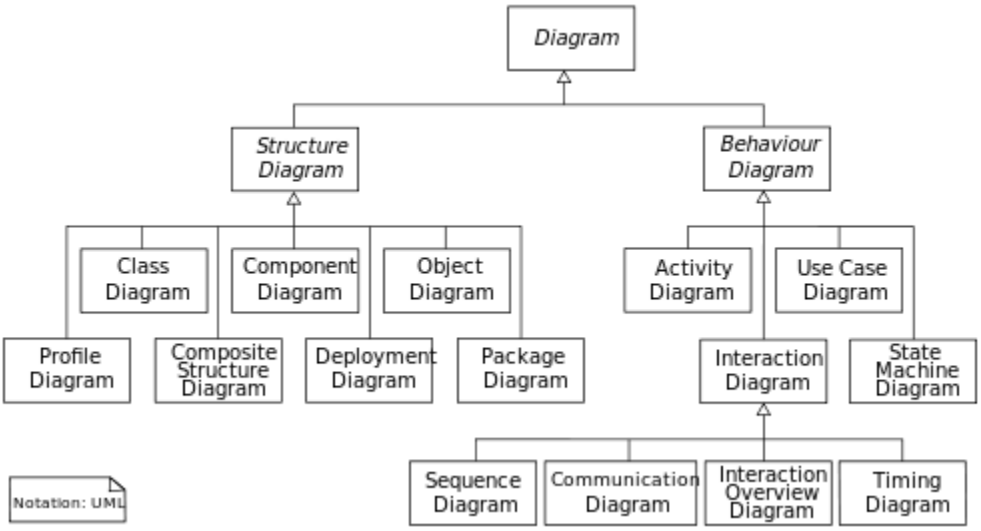
- **Creational:**
  - **Abstract Factory:** Provide an interface for creating families of related or dependent objects without specifying their concrete classes. Would be a an abstract factory class with multiple implementing concrete factory classes.
  - **Factory(Simplified version of Factory Method):** Creates objects without exposing the instantiation logic to the client and Refers to the newly created object through a common interface. Factory method is similar but is a factory with factory subclasses, the first factory can do things with the final product.
  - **Singleton:** Ensure that only one instance of a class is created and Provide a global access point to the object. Useful for loggers, configuration classes, factories which products should always have different id numbers.
- **Behavioral:**
  - **Command:** Encapsulate a request in an object, Allows the parameterization of clients with different requests and Allows saving the requests in a queue.
  - **Observer:** defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and update automatically.
  - **Strategy:** defines a family of algorithms, encapsulates each one, and makes them interchangeable. Strategy lets the algorithm vary independently from clients that use it. (Using an interface with implementing classes).
  - **Template Method:** Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.
  - **Iterator:** Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.
  - **State:** Allow an object to alter its behavior when its internal state changes. The object will appear to change its class. It is similar or a mix of Command and Strategy pattern.
- **Structural:**
  - **Decorator:** Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to sub-classing for extending functionality.
  - **Adapter:** Convert the interface of a class into another interface clients expect. / Adapter lets classes work together, that could not otherwise because of incompatible interfaces.
  - **Facade:** Provides a simplified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to user.
  - **Composite:** Compose objects into tree structures to represent part-whole hierarchies. / Composite lets clients treat individual objects and compositions of objects uniformly.
  - **Proxy:** Provide a “Placeholder” for an object to control references to it. Helps to avoid creating an expensive object until completely necessary.

Programming Practices

- Programming by contract sets up an agreement about how your software behaves that you and users of your software agree to abide by.
- Defensive programming doesn't trust other software, and does extensive error and data checking to ensure the other software doesn't give you bad or unsafe information.

UML (Unified Modeling Language)

Diagrams: https://en.wikipedia.org/wiki/Unified_Modeling_Language



SOA Analysis and Design

Design Principles

From: http://serviceorientation.com/serviceorientation/index

- **Standardized service contract :** guarantee that the service contracts within a service inventory (enterprise or domain) adhere to the same set of design standards.
  - **Functional Expression Standardization:** Standard names for services operations.
  - **Data Model Standardization.**
  - **Policy Standardization.**
- **Service loose coupling :** to ensure that the service contract is not tightly coupled to the service consumers and to the underlying service logic and implementation. Coupling types:
  - **Contract-to-Logic:** contract first (is preferred).
  - **Logic-to-Contract:** using automated tools to generate contract from the logic (negative coupling).
  - **Contract-to-Implementation:** the contract is designed based on underlying implementation details (negative coupling). Can be corrected through the Service Facade pattern.
  - **Contract-to-Technology:** exposes proprietary elements used in the service (negative coupling).
  - **Contract-to-Functional:** exists when the service contract is developed by keeping a particular kind of consumer in mind. Doesn't matter if the service will not be reused.
  - **Consumer-to-Implementation:** This is a negative form of coupling that exists because the service consumers access the service directly either via its logic or implementation. Implementing Contract Centralization Design Pattern helps avoid this.
  - **Consumer-to-Contract:** This is a favorable type of coupling as it helps to evolve the service without impacting its consumers.
- **Service abstraction :** the service contract should not contain any superfluous information that is not required for its invocation. Abstraction's types:
  - Functional Abstraction.
  - Technology Abstraction.
  - Logic Abstraction.
  - Quality Abstraction.
- **Service reusability**
- **Service autonomy :** provide services with improved independence from their execution environments.
- **Service statelessness** : in order to design scalable services by separating them from their state data whenever possible,
- **Service discoverability :** which emphasizes making services discoverable by adding interpretable meta-data to increase service reuse and decrease the chance of developing services that overlap in function.
- **Service composability :** encourages the design of services that can be reused in multiple solutions that are themselves made up of composed services.

Design Patterns

Based on: http://simplicable.com/new/10-soa-design-patterns-every-architect-should-know

For a complete list: http://soapatterns.org/design_patterns/overview

- **Agnostic Capability:** Agnostic service logic is partitioned into a set of well-defined capabilities that address common concerns not specific to any one problem, increasing reusability.
- **Atomic Service Transaction:** Runtime service activities can be wrapped in a transaction with rollback feature that resets all actions and changes if the parent business task cannot be successfully completed.
- **Enterprise Service Bus:** An ESB acts as a message broker between consumers and services. The ESB can perform message transformations, routing and connect to applications via a variety of communication protocols.
- **Service Façade:** A service façade component is used to abstract a part of the service architecture with negative coupling potential. (**personal example at Indra: queryInvoice, payments, etc)**.
- **Service Callback**: A service requires its consumers to call it asynchronously. If the consumer needs a response it provides a callback address. When the service reaches some milestone in processing it messages the consumer with a response. This approach frees resources and is useful when services are expected to be long running. **(personal example at Indra: billing).**
- **Concurrent Contracts:** Multiple contracts can be created for a single service, each targeted at a specific type of consumer. **(personal example at Indra: when required two interfaces one for iProcess and another one to soa. WHICH SERVICE?).**
- **Canonical Schema:** Data models for common information sets are standardized across service contracts within an inventory boundary. (**personal example at Indra using OAGiS and P2P creating one)**
- **Asynchronous Queuing:** A service can exchange messages with its consumers via an intermediary buffer, allowing service and consumers to process messages independently by remaining temporally decoupled. (**personal example at Indra, billing mostly, sales and payments).**
- **Event-Driven Messaging:** The consumer establishes itself as a subscriber of the service. The service, in turn, automatically issues notifications of relevant events to this and any of its subscribers.

Scalability

Reference: Gayle. 2011. Cracking the Coding Interview. 5Th Edition.

Problems Approach

1. **Make Believe:** How would you solve it without resources limitations?
2. **Get Real:** Divide and identify each of the problems.
3. **Solve problems:** Solve each one of the problems that arise.

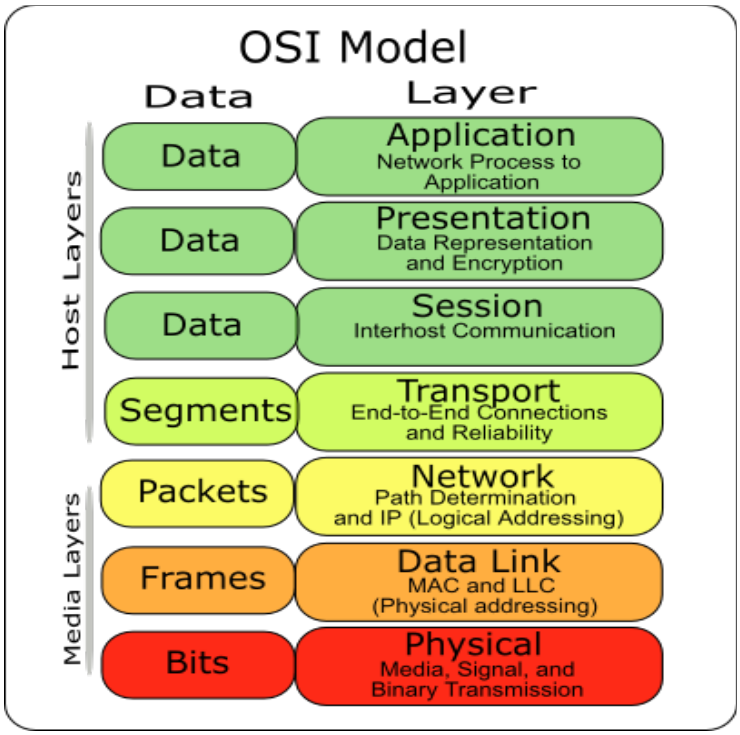
Dividing Lots of Data

- By order of appearance.
- By Hash value.
- By actual value (ex: social networks).
- Arbitrarily.

Networking

OSI (Open Systems Interconnection) Model

https://en.wikipedia.org/wiki/OSI_model



TCP/IP: http://www.w3schools.com/tcpip/tcpip_intro.asp

Cloud Computing

CAP Theorem

It is impossible for a distributed computer system to simultaneously provide: Consistency, Availability and Partition Tolerance. (https://en.wikipedia.org/wiki/CAP_theorem). When checking CAP theorem, usually you say what's the preference under partition. So systems are mostly CP or AP. For example, Cassandra is AP and Hbase is CP.

Concepts

From Cloud Computing Concepts course (Coursera): https://www.coursera.org/learn/cloud-computing

CS Fundamentals review: https://www.coursera.org/learn/cloud-computing/quiz/YRJDz/prerequisite-quiz

- **Gossip:** Randomly message some nodes in the network. It can be pull or Push.
- **Membership:**
  - Membership requires failure detection.
  - Is impossible to have **Completeness (all errors are detected)** and **Accuracy (only errors are detected)** in lossy networks (Chandra and Toueg).
  - Usually completeness is guaranteed.
  - **Centralized heartbeat:** each process sends a heartbeat. Good as long as central node doesn't fail.
  - **Ring heartbeat:** each process heartbeats 2 others. Con: unpredictable on multiple failures.
  - **All to All heartbeat:** equal load per member. Best to detect issues. The problem is one node can be the problem, but mark others as problematics.
  - **Gossip style:** nodes share the heartbeat count among them. Need 2 timeouts before deleting bad node (Tfail and Tcleanup).
- **P2P systems:**
  - **Napster:** Set of servers with information about the peers content. Clients query servers, ping peers and download from best node.
  - **Gnutella:** removed servers.
  - **FastTrack:** Similar to Gnutella, but with "supernodes" (do more things based in reputation).
  - **BitTorrent:** Get Tracker (.torrent, address). Tracker has information of nodes transmitting that file currently.
    - File split into blocks (32-256KB)
    - Downloads Local Rarest First
    - Tit for tat: prefers to give to nodes that you downloaded from with best download rates
    - Choking: Limit number of neighbors only to best.
- **Time and Ordering:**
  - Need to keep clocks in sync (Clock Skew vs Clock Drift)
  - How often to sync? Check Maximum Drift Rate (MDR).
  - There's external (Cristian's Algorithm and NTP) and internal syncronization (Berkeley Algorithm).
  - External Synchronization with bound D means internal synchronization with a max of 2*D. But internal

synchronization may make the group drift from external clock.

- **Cristian's Algorithm:**
  - Bounds the time that takes to ask for the time and receiving the time from an external source, and uses it to set the current time.
  - Gotchas:
    - Cannot decrease clock time. -> to prevent inconsistencies.
    - Can increase or decrease speed of clock.
    - Can avg multiple reads.
- **NTP:**
  - Organized in trees.
  - Asks for the time twice, stores server(parent) times (ts1, ts2) and client time (tc1, tc2) and uses them to get the offset.
- **Lamport timestamps:** Used by most cloud computing systems.
  - Happens-before or causality.
  - When sending events, send a counter and increase it.
  - If there's no causal path between 2 events, they are concurrent. (not very precise)
- **Vector Clocks:** Similar but with vector carrying the counter for each process, so more precise.

- **Snapshots:**
  - Global snapshot: state for each process and state for each channel.
  - There are algorithms and stuff to get it. (Chandy-Lamport)
  - Consistent cut => obeys causality.
  - **Safety and liveness:**
    - Help check correctness of distributed systems.
    - **Liveness:** guarantee that something good will happen eventually. (eventually a failure will be detected, a consensus will be reached).
    - **Safety:** guarantee that something bad will never happen. (no object is orphaned, no deadlocks, accuracy in failure detection)
    - Not always possible:
      - Failure detector
      - Consensus decisions
    - All stable (once true, always true) global properties can be detected by Chandy-Lamport.
- **Multicast:** Ordering: Fifo, Causal, Total. Examples: social network comments. Needs to be causal.
- **Virtual synchrony or view synchrony:**
  - Attempts to preserve multicast ordering and reliability in spite of failures.
  - Combines a membership protocol with a multicast protocol.

Paxos

https://en.wikipedia.org/wiki/Paxos_(computer_science)

Family of protocols to address consensus problems (not always possible to solve). For example, a group of servers attempting:

- Make sure that all of them receive the same updates in the same order as each other (Reliable Multicast).
- Keep their own local lists where they know about each other, and when anyone leaves or fails, everyone is updated simultaneously (Membership/ Failure Detection).
- Elect a leader among them, and let everyone in the group know about it (Leader Election).
- To ensure mutually exclusive (one process at a time only) access to a critical resource like a file (Mutual Exclusion).

Distributed systems can be synchronous or asynchronous. The consensus problem is solvable in the former and impossible in the latter. Paxos (invented by Lamport) solves it in asynchronous systems.

In theory doesn't guaranteed getting to a consensus, but in practice it reaches consensus very quickly.

Front End

- How browsers work: http://www.html5rocks.com/en/tutorials/internals/howbrowserswork/
  - HTML is not context-free grammar (cannot be parsed as XML).
  - Scripts are executed synchronously while parsing. If want to defer can use “defer” attribute.
  - SSL encrypts on transport layer.
- What happens when you browse a website in your browser? http://superuser.com/questions/31468/what-exactly-happens-when-you-browse-a-website-in-your-browser

- How to improve website performance (https://developer.yahoo.com/performance/rules.html):
  1. Minimize HTTP requests: combine files, minify, CSS sprites.
  2. Use a CDN.
  3. Add an Expires or a Cache-Control Header.
  4. Gzip components.
  5. Put stylesheets at the top (allows the page to load progressively).
  6. Put scripts at the bottom: Http1.1 establishes no more than 2 downloads in parallel from same hostname. While a script is downloading the browser won't perform other requests. Use defer or async.
  7. Avoid CSS expressions.
  8. Make Javascript and CSS external: reduces HTML size and can be cached (HTML not).
  9. Reduce DNS lookups.
  10. Minify JS and CSS.
  11. Avoid redirects.
  12. Remove duplicate scripts.
  13. Configure Etags (like MD5).
  14. Make AJAX cacheable.
  15. Flush the buffer early.
  16. Use GET for AJAX requests (if you are not posting data).
  17. Post-load components.
  18. Preload components (Unconditional, Conditional or Anticipated).
  19. Reduce the number of DOM elements.
  20. Split components across domains.
  21. Minimize the number of iframes.
  22. No 404s.
  23. Reduce cookie size.
  24. Use Cookie-free domains for components (make static components cookie-free requests).
  25. Minimize DOM access (cache references to elements, update nodes “offline” and make one update, avoid fixing layout with JS).
  26. Develop smart event handlers (if you have a div with 10 buttons, handle the event on the div instead of each button).
  27. Choose <link> over @import (@import behaves as a link at the bottom of the page).
  28. Avoid filters.
  29. Optimize images (check palette, convert gifs to pngs).
  30. Optimize CSS sprites.
  31. Don't scale images in HTML.
  32. Make favicon.ico small and cacheable (it is always requested, and cookies are sent on each request).
  33. Keep components under 25K (iPhone won't cache components bigger than 25K).
  34. Pack components into a multipart document.
  35. Avoid empty image src.

- Web accessibility initiative (WAI: http://www.w3.org/WAI/intro/accessibility.php):
  - Text alternative for non-text content.
  - Adaptable (content presented in different ways).
  - Keyboard accessible.
  - Enough time.
  - Navigable.
  - Input assistance.
  - Compatible.
- CSS3 major improvements (http://tutorialzine.com/2013/10/12-awesome-css3-features-you-can-finally-use/):
  - Animations and transitions.
  - Calculating value with Calc.
  - Advanced selectors.
  - Generate content and counters.
  - Gradients.
  - Webfonts.
  - Box sizing.
  - Border images.
  - Media queries.
  - Multiple backgrounds.
  - CSS Columns.
  - CSS 3D transforms.
- HTML5 major improvements (http://www.developer.com/lang/5-html5-features-every-developer-should-know-how-to-use.html):
  - Multimedia elements (audio, video).
  - Canvas element (API to render drawings).
  - New input types (tel, email, number, color, url, range, etc).
  - Miscellaneous Form Features (mark fields required, autofocus, watermark, regex).
  - Custom data attributes (won't affect UI).
  - Others (http://blog.elemdage.com/technology/the-7-most-important-html5-features)
    - Web Workers
    - .Geolocation.
- Responsive Web Design (RWD):
  - https://developers.google.com/web/fundamentals/layouts/rwd-fundamentals/
  - http://en.wikipedia.org/wiki/Responsive_web_design

  Responsive web design (**RWD**) is an approach to web design aimed at crafting sites to provide an optimal viewing experience—easy reading and navigation with a minimum of resizing, panning, and scrolling—across a wide range of devices (from desktop computer monitors to mobile phones).[1][2][3]

  A site designed with RWD[1][4] adapts the layout to the viewing environment by using fluid, proportion-based grids,[5][6] flexible images,[7][8][9] and CSS3 media queries,[3][10][11] an extension of the `@media` rule, in the following ways:[12]

  - The fluid grid concept calls for page element sizing to be in relative units like percentages, rather than absolute units like pixels or points.[6]
  - Flexible images are also sized in relative units, so as to prevent them from displaying outside their containing element.[7]
  - Media queries allow the page to use different CSS style rules based on characteristics of the device the site is being displayed on, most commonly the width of the browser.

Javascript

https://www.interviewcake.com/question/js-whats-wrong

- “When a function accesses a variable outside its scope, it accesses that variable, not a frozen copy”
- Event propagation JS:
  - Capturing (addEventListener with thir parameter == true), Event Handlers to specific objects, Event bubbling on DOM elements.

Other

What's a framework?

From: https://en.wikipedia.org/wiki/Software_framework

It should have:

- **inversion of control** – In a framework, unlike in libraries or normal user applications, the overall program's flow of control is not dictated by the caller, but by the framework.[1]
- **default behavior** – A framework has a default behavior. This default behavior must be some useful behavior and not a series of no-ops.
- **extensibility** – A framework can be extended by the user usually by selective overriding or specialized by user code to provide specific functionality.
- **non-modifiable framework code** – The framework code, in general, is not allowed to be modified, excepting extensibility. Users can extend the framework, but not modify its code.

Brain Teasers

- Try to write the assumptions you're making. Try to change them, one by one.
- Obvious answers probably not the correct answer.

Other

- 32 bits vs 64 bits (Mongan, Giguere, Kindler. 2012. Programming Interviews Exposed: Secrets to Landing your Next Job. Page 247)
- Hash tables are more efficient to do operations, but trees are more efficient in memory to keep things ordered (ex: contacts in mobile device).
- How to create cookies (JS): http://www.w3schools.com/js/js_cookies.asp

Quality Assurance

- ISO-9126: Standard for Software Quality evaluation. It specifies the following dimensions:
  - Functionality
  - Reliability
  - Usability
  - Efficiency
  - Maintainability
  - Portability
- MOSCA: stands for Systemic Quality Model (in Spanish), a quality model developed by Universidad Simón Bolívar, based on: ISO-9126, ISO-15504, CMM y PSP,

GUI Design

- Defined in standard ISO 9241 (http://en.wikipedia.org/wiki/User_interface_design):
  - Suitability for the task.
  - Self-descriptiveness.
  - Controllability.
  - Conformity with user expectations.
  - Error tolerance.
  - Suitability for individualization.
  - Suitability for learning.

Reactive Systems

http://www.reactivemanifesto.org/

Systems should be:

- Responsive
- Resillient
- Elastic
- Message Driven

Databases

Translating Entity Relationship Diagram (ERD) to Relational

From: Ramez Elmarsi, Shamkat B. Navathe . Fundamentals of Database Systems, 3rd Edition.

1. For each regular (strong) entity E is created a relation R that contains all simple attributes of E, including all simple attributes of an composed attribute. Primary key is kept.

2. For each weak entity W with propietary entity E, is created a relation R and are included all the simple attributes of W in R. R also includes the primary key of all the proprietary entities. R's primary key is the combination of all the primary keys and partial key of W.

3. For each 1:1 R relation in ERD, are identified relations S and T that correspond to the entity types that participate in R. Select one of this relations, say S, and include as S' external primary key T's primary key. All simple attributes from 1:1 R relation should be included in S.

4. For each 1:N binary relation R, is identified S relation that represents the entity with N participation. Include T's (the other relation) primary key as S foreign key. All R attributes are included in S.

5. For each M:N binary relation R, is create a new S relation to represent R. Primary keys from participating entities are included as S foreign keys, the mix of this keys will be S' primary key. All attributes from R are included in S.

6. For each multi-valued attribute A is create a new R relation. This will include A attribute and the primary key K (as foreign key) from the entity E that has A as attribute. R's primary key is A and K. If A is composed, then are included its attributes.

7. For each n-ary relation R, n>2 is created a new relation S that represents R. All primary keys from participating entities are included as S' foreign key. R's simple attributes are also included in S. S' primary key usually is a combination of the foreign keys (excluding the attributes from entities with participation in R = 1).

Additional rules apply for Extended Entity Relationship Diagram.

Normalization

According to Wikipedia: https://en.wikipedia.org/wiki/Database_normalization

- **1NF:** A relation is in first normal form if the domain of each attribute contains only atomic values, and the value of each attribute contains only a single value from that domain.
- **2NF:** No non-prime attribute in the table is functionally dependent on a proper subset of any candidate key.
- **3NF:** Every non-prime attribute is non-transitively dependent on every candidate key in the table. The attributes that do not contribute to the description of the primary key are removed from the table. In other words, no transitive dependency is allowed.

There are other Normal Forms.

Transactions

Transactions must satisfy ACID: Atomicity, Consistency, Isolation, Durability.

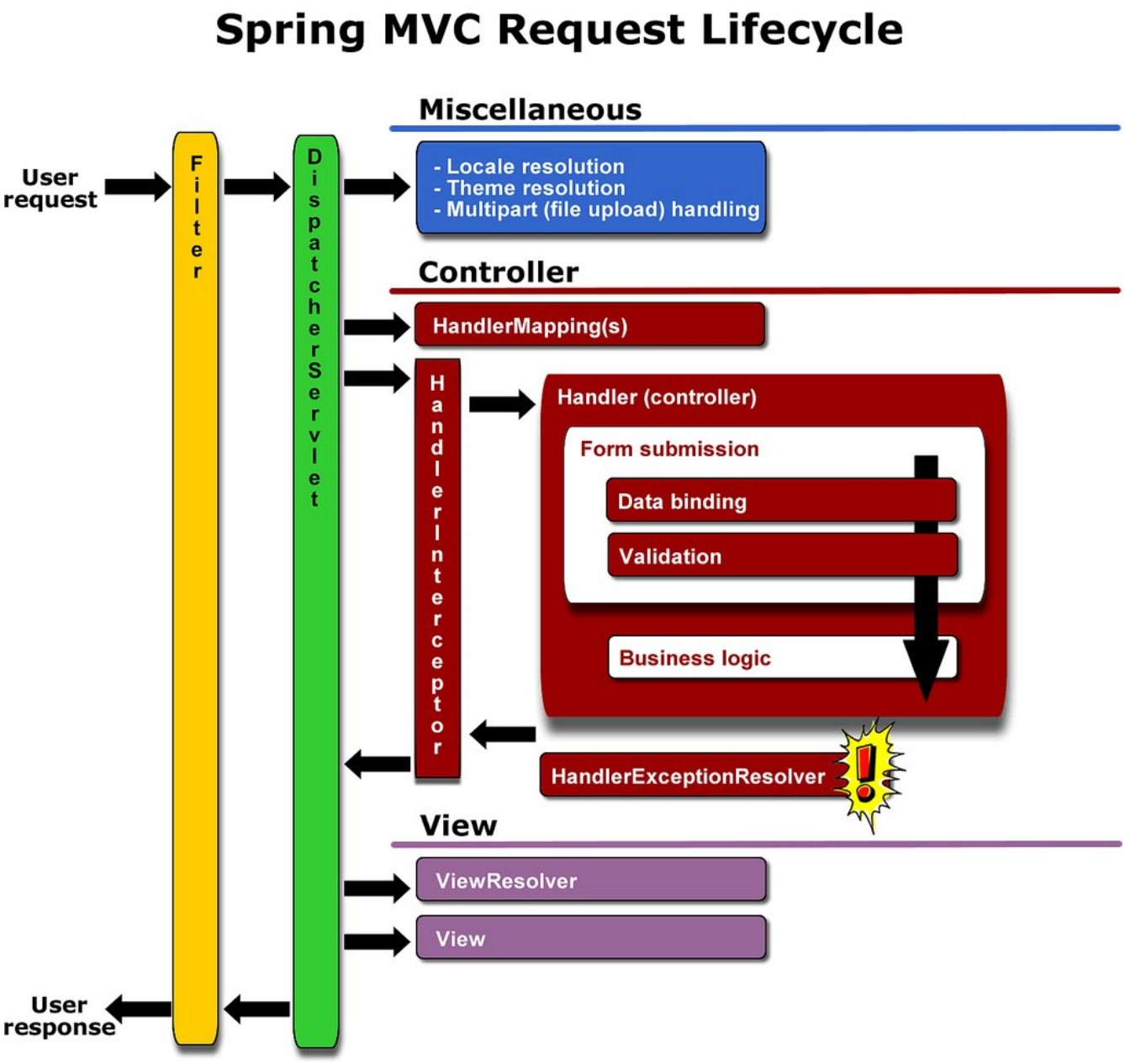
Spring MVC

- What is? http://en.wikipedia.org/wiki/Spring_Framework

- IoC: http://docs.spring.io/spring/docs/current/spring-framework-reference/html/beans.html

- AOP: http://docs.spring.io/spring/docs/current/spring-framework-reference/html/aop.html

Request lifecycle: https://www.flickr.com/photos/60896767@N00/89101625/sizes/l/



Web Services in Java

References for this section are from:

- Kalin. 2013. Java Web Services Up and Running
- Hewitt. 2009. Java SOA Cookbook

SOAP Web Services (Using JAX-WS)

TIP: SOAP stood for Simple Object Access Protocol.

Web Service Creation

Implement a Service Endpoint Interface (SEI) using @WebService annotation (additional annotations help customizing it)

```
@WebService(targetNamespace="http://www.librethinking.com/schemas/awsschemas")
@SOAPBinding(style = SOAPBinding.Style.DOCUMENT, use=SOAPBinding.Use.LITERAL)
//default
public interface CheckUserSEI {
    @WebMethod(operationName = "CheckUserRqRpOp")
    @RequestWrapper(localName="CheckUserRequest")
    @ResponseWrapper(localName="CheckUserResponse")
    @WebResult(name="CheckUser",targetNamespace="http://...awsschemas")
    public CheckUsersMessage CheckUser(
            @WebParam(name="CheckUser", targetNamespace="http://...awsschemas")
            CheckUsersMessage input);
}
```

Then we implement the Service Implementation Bean (SIB) using @WebService(endpointInterface=”myinterface”) and implementing the SEI. The annotation endpointInterface makes all the annotations of the interface to be honored instead of the annotations in the implementing class.

```
@WebService(serviceName = "CheckUser",
        endpointInterface = "package.CheckUserSEI")
public class CheckUserSIB implements CheckUserSEI{
    @Override
    public CheckUsersMessage CheckUser(CheckUsersMessage input) {...}
}
```

If the SOAP binding is Document (default) then its necessary to execute the next statement in order to create additional files necessary to generate the WSDL:

`% wsgen -keep -cp . package.CheckUserSIB` (my service)

Generate Client

To generate a client first is needed to create the support code using wsimport:

```
wsimport -keep -p myClient url_to_wsdl
```

Depending on the WSDL top element, by default the generated client's artifacts will be wrapped. That means one top element that must have all the subelements set before making the call. For example, the state above, creates something like this:

```
@SOAPBinding(parameterStyle = SOAPBinding.ParameterStyle.WRAPPED)
public interface IMathServer {
    @WebMethod
    @WebResult(name = "addNumsResponse")
    public int addNums(@WebParam(name = "num1") int num1, @WebParam(name = "num2")
int num2);
}
```

According to JavaDocs SOAPBindings (http://docs.oracle.com/javaee/5/api/javax/jws/soap/SOAPBinding.html):

Determines whether method parameters represent the entire message body, or whether the parameters are elements wrapped inside a top-level element named after the operation.

To generate an unwrapped client artifacts can be executed the following:

```
wsimport -keep -p myClient url_to_wsdl -b custom.xml
```

Where custom.xml has the following:

```
<jaxws:bindings
      wsdlLocation = "url_to_wsdl"
      xmlns:jaxws="http://java.sun.com/xml/ns/jaxws">
<jaxws:enableWrapperStyle>false</jaxws:enableWrapperStyle>
</jaxws:bindings>
```

More on this here: http://stackoverflow.com/questions/5324051/webservices-bare-vs-wrapped/18344156#18344156

To call the web service (I.e: create the client) after the artifacts have been generated the following can be done:

```
  YourServiceClass service = new YourServiceClass();
  YourEndpointClass port = service.getPort();
  YourRequestClass request = new YourRequestClass();
  YourMessageClass message = new YourMessageClass(); //In case you have it
  message.setParam1(param1);
  message.setParam2(param2);

  request.setMessage(message);
  YourResponseClass response = port.ServiceOperation(request); //This call locks
  System.out.println(response.getMessage().getResponse());
```

YourServiceClass is the generated artifact the extends javax.xml.ws.Service.
YourEndpointClass can be seen in YourServiceClass in an operation that calls super.getPort();
YourRequestClass and YourResponseClass will depend on how is managed the Request and Response message.
YourMessageClass would be a wrapper class for your message (depending on WSDL)

In case the client artifacts where Wrapped, the example would possibly have the method port.ServiceOperation() with more parameters that will later be *wrapped* into a single top-level element. With Bare artifacts there would be no difference.

Other

Wrapped vs Unwrapped

Wrapped is nicer and most used. One big xml element with sub-elements instead of many xml element on the same level.

Document vs RPC Style

Document has richer types. RPC only basic.

Getting Access to SOAP Message

By default JAX-WS hides the SOAP message details, but if it is necessary to perform Protocol-Specific Work you can implement the SOAPHandler interface and set it to the Web Service like this:

```
service.setHandlerResolver(new HelloHandlerResolver());
```

(Necessary, for example, to add authentication when connecting to AWS).

Asynchronous Client

To make the client asynchronous the client artifacts must be generated again with a custom.xml with:

<jaxws:enableAsyncMapping>true</jaxws:enableAsyncMapping>

Then the call will require a Handler (implementing javax.xml.ws.AsyncHandler).

Create Java Classes from Schema

If you have an xsd you can create the required Java classes with xjc:

```
xjc –verbose –d Schema.xsd
```

SAAJ

SOAP with Attachments API for Java.

RESTful Web Services

REpresentation State Transfer.

Using JAX-WS

Using @WebServiceProvider annotation and implementing Provider (to avoid JAXB).

(Hewitt. 2009. Java SOA Cookbook)

```
@WebServiceProvider
@BindingType(HTTPBinding.HTTP_BINDING) //Instead of SOAP
public class JaxWsRestfulService implements Provider<Source> {
      @Resource
      protected WebServiceContext ws_ctx; //dependency injection
      public Source invoke(Source request) {
            MessageContext msg_ctx = ws_ctx.getMessageContext();
      String http_verb = (String) msg_ctx.get(MessageContext.HTTP_REQUEST_METHOD);
            if(http_verb.trim().toUpperCase().equals("GET")) return doGet(msg_ctx);
            …
            ByteArrayInputStream stream = …;
            return new StreamSource(stream);
      }
      public Source doGet(MessageContext msg_ctx) {...}
}
```

A client can be implemented using sockets. For reference check: Hewitt. 2009. Java SOA Cookbook. Section 8.3.

Using JAX-RS (Jersey)

The easiest standard way to develop. Import Jersey into the project. Modify wb.xml including as servlet-class:

```
com.sun.jersey.spi.container.servlet.ServletContainer
```

Use the annotations:

- @Path: to specify the path to which the service should respond. This can include regular expressions as well as parameters (ex: `@Path("/products/{id: \\d{3}}")`)
- Annotate methods with @GET, @POST, @PUT, @DELETE and @HEAD. You can also define multiple times this annotations and using @Produces annotation to specify which one should be used. If @Produces is not specified then it assumes it can handle everything. If @Produces is specified but when a call is received can't find an adequate method, then error 406 is returned (http://docs.oracle.com/javaee/6/api/javax/ws/rs/Produces.html).
- In methods can be used @PathParam to identify the expected param for each variable.

Example:

```
@Path("/hello/{user}")
public class CheckUserREST {
    @GET
    @Produces("text/html")
    public String getUser(@PathParam("user") String user) {
        return "<html><body><h1>Hello "+user+"!</h1></body></html>";
    }
}
```

Client can be also be generated with sockets.

Using Servlets

Implementing the operations doGet, doPost, doDelete, doPut, etc from the Servlet. As it is necessary to manipulate the XML, JAXB or SAAJ would probably be required.

Using Restlet Framework

More info here: http://restlet.org/

Servlets

(using @WebServlet in new servlets //TODO)

SOAP vs REST

REST is an architectural pattern and SOAP is a protocol.

From: http://spf13.com/post/soap-vs-rest

REST

RESTs sweet spot is when you are exposing a public API over the Internet to handle CRUD operations on data. REST is focused on accessing named resources through a single consistent interface.

SOAP

SOAP brings it's own protocol and focuses on exposing pieces of application logic (not data) as services. SOAP exposes operations. SOAP is focused on accessing named operations, each implement some business logic through different interfaces.

Why REST?

- Uses standard HTTP so it is much simpler in just about every way.
- REST permits many different data formats where as SOAP only permits XML.
- REST has better performance and scalability. REST reads can be cached.

Why SOAP?

- WS-Security: While SOAP supports SSL (just like REST) it also supports WS-Security which adds some enterprise security features.
- WS-AtomicTransaction: Supports ACID Transactions.
- WS-ReliableMessaging: SOAP has successful/retry logic built in and provides end-to-end reliability even through SOAP intermediaries.

Other sources:

http://blog.smartbear.com/apis/understanding-soap-and-rest-basics/

http://stackoverflow.com/questions/19884295/soap-vs-rest-differences

Cannot fully build a SOA with REST:

http://stackoverflow.com/questions/10491812/can-a-soa-be-designed-with-rest