*RMM – The steps how to make REST API mature to make it easy to understand, easy to navigate for user.*

*Three imp factors –*

1. *URI – There must be one URI per resources.*
2. *HTTP – Making use of HTTP, and proper Methods (Get, Post, Put…)*
3. *HATEOAS – No need to verify any document to interact with REST. The interaction should happn through the server’s response.*

*Level 0 – At Level 0. Single URL and sameHTTP method for single resource.*

*Example – add user – POST* [*http://localhost:8080/user*](http://localhost:8080/user) *get user – POST* [*http://localhost:8080/user*](http://localhost:8080/user)

*Level 1 – different resources, different URL but use POST only*

*Example – add employee – POST* [*http://localhost:8080/department/{dept-id}/e*](http://localhost:8080/department/%7bdept-id%7d/e)*mployee get user – POST* [*http://localhost:8080/department/{dept-id}/employee/{employee*](http://localhost:8080/department/%7bdept-id%7d/employee/%7bemployee) *-id}*

*Level 2 – we are using appropriate HTTP method and getting correct response. Like 200 OK, 500 error etc*

*Example – add user – POST* [*http://localhost:8080/user*](http://localhost:8080/user) *get user – GET* [*http://localhost:8080/user*](http://localhost:8080/user)

*Level 3 – should implement HATEOAS to help user to navigate where new resources can be found.*

*Test Pyramid – This is one of the most important testing strategies that help developers and testers to make a high-quality, cost-effective product.*

*Three levels –*

*Unit test – Needs to write by developer. Help to identify issues in code at very early stage. This is to test how each unit of code like functionality, method working independently. Any external resources like DB, Web Services, APIs to be mock. Generally, use negative scenarios, write separate test for separate scenarios with separate test data. Name should match to the purpose of the test.*

*Integration testing – In this we check, how interaction between two codes, functionalities is working. The interaction can be between code and DB, with another web services, with APIs etc. This should be done by QA team, automation will be effective, after development is done, test both positive and negative scenarios. It is a bit time consuming.*

*E2E testing – End 2 End testing is most important and time-consuming test strategy where we actually test in a way user can look into the application. In this, we have actual interaction between all components. We test the product start to end like from user interface to back-end. QA should perform this, should cover all scenarios. Should perform after integration testing. It is crucial and time taking hence kept at apex in pyramid.*

*Agile and test pyramid – This pyramid is useful for agile team, as it helps to create quality, cost effective product. We can launch product in market early can get feedback early from user. Helps developer to get issues in a code at early stage. This helps to write code simple, reliable. Helps team to prioritize task, work. Team can manage time and achieve effective results. Can identify risk.*

*FIRST – This is unit testing principle.*

*Fast – Developer can run test at any phase of development. It should run fast even though there are thousands of errors.*

*Independent/Isolated – Each test case should be independent of each other. It must have separate test data and results generated should not be dependent on any other things like environment, variables etc. 3A’s –*

* *Arrange – data should provide for each test case.*
* *Act – should call actual method.*
* *Assert – should act one logical assert, multiple asserts should use for single state of object. Nothing should be written after assert.*

*Repeatable – Each test can run repeatedly and should not affect result in any environment. Should not depend on any external factors.*

*Self-validating – Test should generate result automatically. No need to check manually if it is passed or fail.*

*Through – should cover happy scenario, cover all corner cases, illegal args and variables, test for security, should cover all use cases not only 100% coverage.*

*Technical Debt – It is a concept is which we omit or delay some part of project work just because of fast delivery and fast completion of project but due to which we may face issues later. It is generally occurring due to sacrifice in system design by developers and other team members.*

1. *Known Debt – Where teams knowingly sacrifice the system design and do imperfect implementation. Teams even could not go back again improve it.*
2. *Unintentional Debt – This occurs if team is not aware of what is requirement currently in the market and team could not make an architecture for the requirement. This happens if developers are unexperienced.*
3. *Unavoidable debt – These are debts occurred due to change of technology or any kind of major change. This is not under any one’s control not even organization. Example upgradation to system, change in programming language. Supporting new devices etc.*

*Causes – lack of knowledge, deadline pressure, change in technology, lack of code review, outdated technology, and temporary solution.*

*Resolving – Regular code review, automate testing, code refactoring, use of modular code and design pattern, technology upgradation, collective code ownership, avoid over engineering, measure debt using code metrics, code churn, time to market, documentation debt.*

*Design Principles – Software design principles helps us to write systematic code, which is easy to understand, easy to scale, support fault tolerance, reliable, low-risk.*

1. *SOLID – S - Class should have only one reason to change.*

*O – Open for extension, close for modification.*

*L – Liskov Substitution*

*I – Interface Separation*

*D – Dependency injection.*

1. *DRY – Don’t Repeat Yourself – Once code written, don’t repeat same code.*
2. *KISS – Keep it, Simple Stupid – Try to make each piece of Software simple and avoid unnecessary complexity.*
3. *YAGNI – You Are Not Gonna Doing It – Don’t write unnecessary code, if not required.*

*Design Patterns – Design Patterns are simple or general solutions for the commonly occurring and repeating problem. As it is tested and proven, they speed up implementation.*

*Pros – Save time, Save money, provide proven solution so no need to think, reusability.*

*Cons – taking decision to use pattern is tough, less experience developers can not implement properly, certain level of expertise is needed.*

*Types –*

* *Creational – Singleton, Prototype, Builder, Factory*
* *Structural – Adapter, Flyweight, Facade, Bridge, Proxy, Decorator*
* *Behavioural – Chain of Responsibility, Strategy, Memento*

*CI/CD Pipeline – Pipeline is a set of steps that needs to perform by developers before delivering a new version of software. These are mandatory set of steps, which were done manually and causes chances of errors, waste of time and less productivity.*

*Automated pipelines remove manual errors, provide standardized feedback loops to developers, and enable fast product iterations.*

*Stages – source stage – code get committed in repository and pipeline get a notification to start. Developer can also start a pipeline manually.*

*Build stage – All dependencies are getting added for source code and building of source code started. It includes compiling, validating etc. Cloud based are build using Docker. Failure in build means issues in software source code.*

*Test Stage – All the test cases created runs in this. Its developer’s responsibility to write test case. This stage takes seconds to hours based on number of test cases written.*

*Deploy stage – In this stage, the build code can be deployed to different environments.*

*Issues at any stage stops the processing and notification send to respective members via email or slack etc.*

*Code Review – Benefits - Increase Collaboration, Better Estimation, Maintain Compliance, Enhance Quality, Reduce Bugs, Sharing Knowledge.*

*Issues – Delays in release, Conflicts, Distribution of task, Longer Review time, Shift Responsibility, No coding standard.*

*Deciding SDLC –*

1. *Project scope and complexity – how well goals are set, how clear design and approach is? How much risk is involved? If these factors are clear and well known in advance go for waterfall, if its dynamic and unclear go for agile.*
2. *Time and Budget - How urgent and flexible are the deadlines and deliverables? How much resources and funding are available for the development process? if your project has tight and fixed deadlines, you may choose the iterative model. if your project has more relaxed and variable deadlines, you may choose the spiral model.*
3. *Project team and communication – Project team geography, communication frequency. Like possible communication go for Agile and different geography go for waterfall.*
4. *Project quality and customer satisfaction - How high are the standards and expectations of the software product? How important is the feedback and involvement of the end-users?*

*T-shirt sizing is a technique used to estimate the size or effort required to complete user stories or tasks. It involves assigning sizes based on T-shirt sizes, like XS, S, M, L, XL, or using the Fibonacci sequence (1, 2, 3, 5, 8, etc.). The story points are in Fibonacci It’s because numbers that are too close to one another are impossible to distinguish as estimates.*

*Propagation Levels in Transaction Management –*

*REQUIRED - REQUIRED is the default propagation type in Spring, and it includes all other transactional methods that are called after it. This means that it is required to create one session for all transactions that are called after it.*