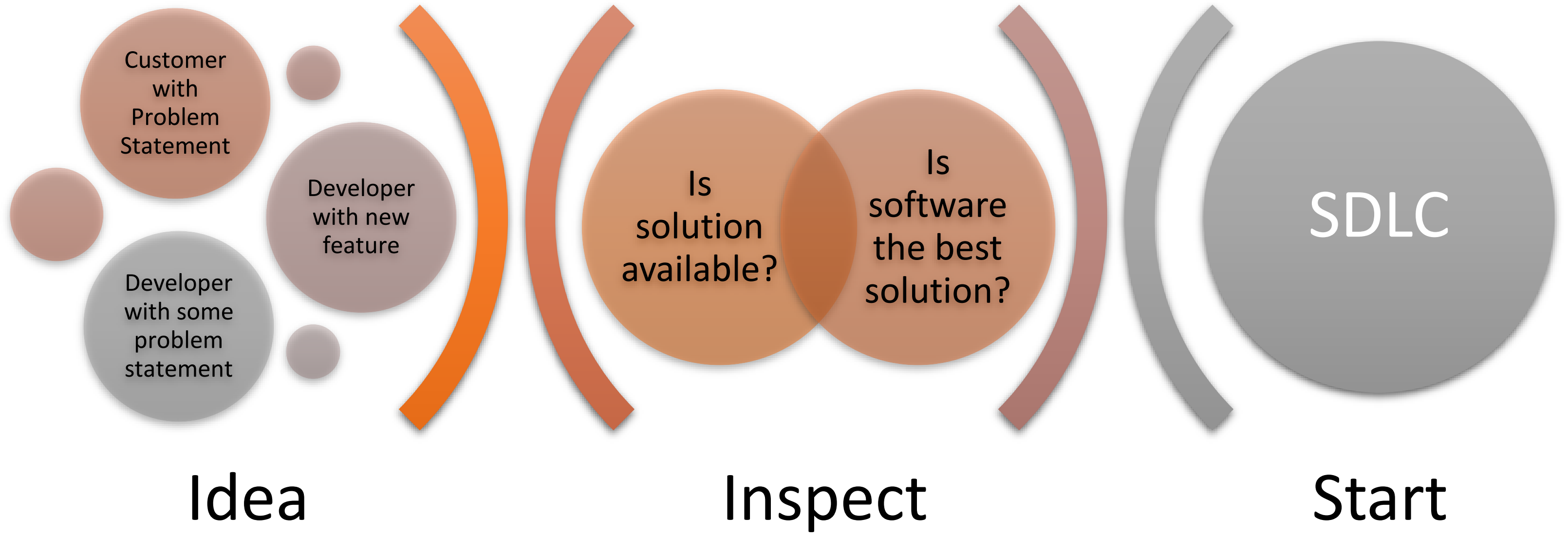
More about application development. Focus on efficiency

Pavankumar Gopalbhai GUPTA PavankumarGopalbhai.Gupta@ca.com

Senior Software Engineer at CA Technologies

November 2018

Session MM


Customer with Problem Statement
Developer with new feature
Developer with some problem statement
Is solution available?
Is software the best solution?
SDLC
Idea
Inspect
Start

- The software is the core of any solution. Today, we have many applications to simplify our day to day tasks. SDLC cycle starts as soon as the software is identified as the potential solution for any idea.
- Not all software last long, and in the competitive environment each problem have many solutions. In this competitive market we only need to be very proactive with our software features and security since it is hard to retain your customer and no one wants to lose the customer.

Today’s session

Product developed need to be of high-quality, but at a predetermined cost and within a predetermined timescale.

- Developing the efficient application.
- Efficiently developing the application.

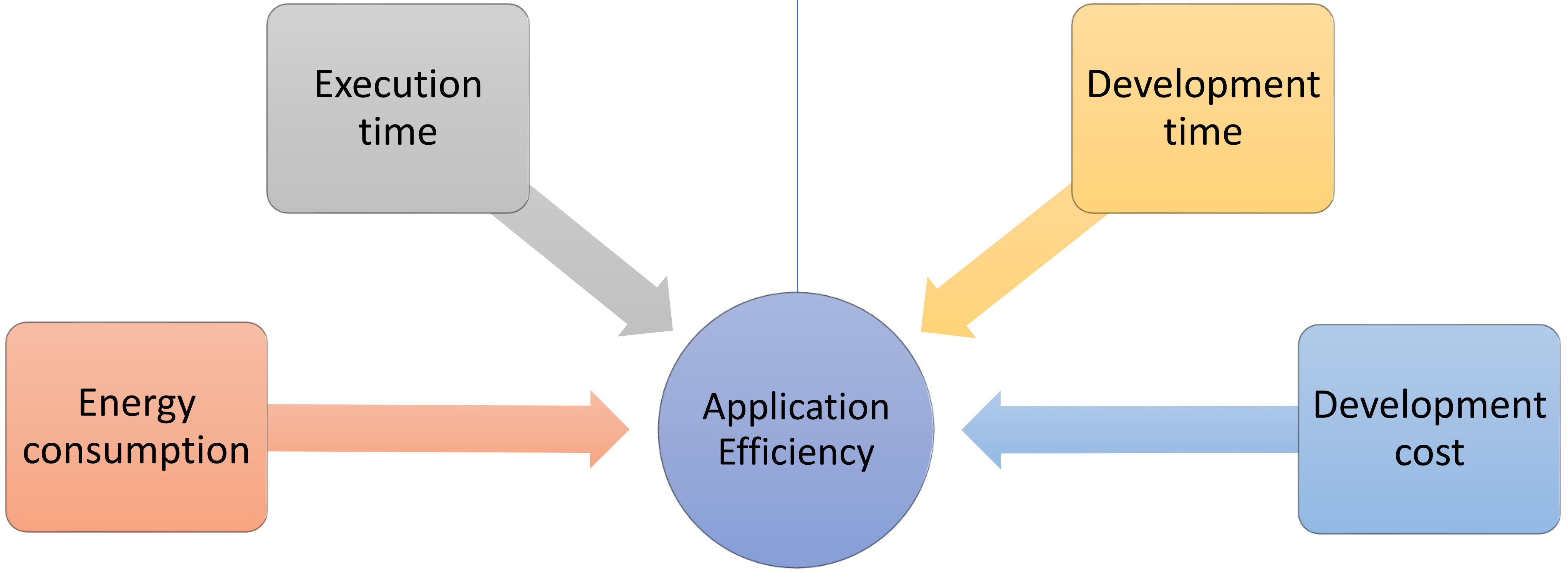


Table of Content

Efficiency overview

Development cost and Implementation time

- Being first is critical to any business application because nothing is perfect on the first implementation and the faster one can get users engaged in the application, the faster you can iterate to make the application better for everyone.
- To reduce implementation time, the approach most organization choose is bringing more number of developers, which increases the cost of development. Adding More Resources Doesn’t Necessarily Speed Up the Work.
- Development costs are the costs a business incurs from researching, growing and introducing a new product or service. The first thing that comes to mind when it concerns cost savings is to hire the cheapest development agency or cheap developers. But then it could a compromise to the quality of the software. Higher development cost directly affect the sales of the application resulting many customers to switch to a cheaper solution.

Application performance

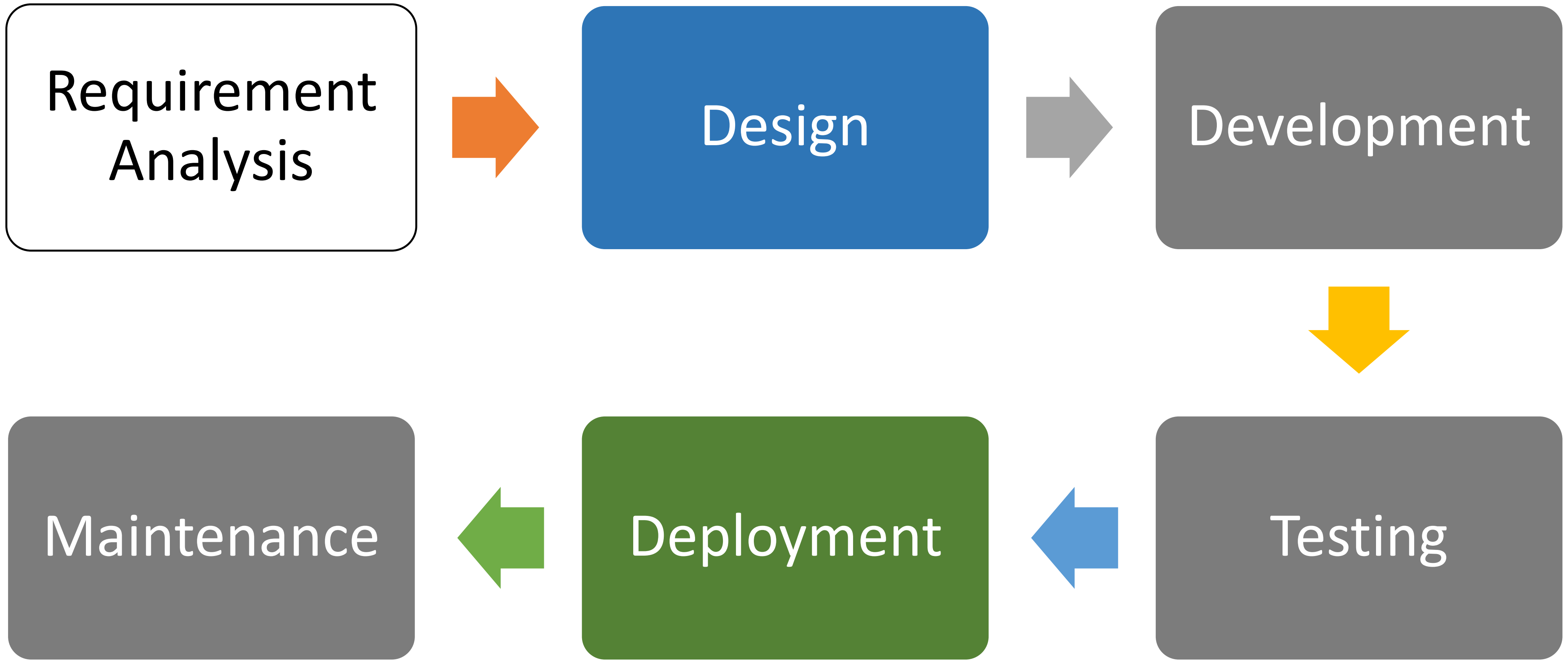
- Even small inefficiencies in apps add up across the system, significantly affecting battery life, performance, responsiveness, and temperature.
- The rapid design and deployment of applications has been mainly done sometimes in the absence of performance considerations and it affects WCET. The worst-case execution time (WCET) of a computational task is the maximum length of time the task could take to execute on a specific platform.
- Application performance is difficult to quantify because it depends on many variables like hardware used, number of concurrent users, database size.
- If you are developing a web application, when it grow and gain popularity, more and more people access your application at the same time. If you don't manage it well, the network and your services will be overloaded (bottleneck) and unable to serve properly.
- Another factor that impact application performance is having too many features, making the application more complex and heavy.

Application energy

- Optimizing performance does not always help to save energy. The common approach to improve application performance is to increase hardware which indirectly increases energy consumption.
- Energy consumption E = power dissipation P over time t, that is, $E = P \times t$.
- For the application running on big machines/servers, an Inefficient application can add the server running and cooling cost.
- For the application running on battery with the device limited capacity, the extensive use of apps that frequently send and receive data to different servers over the network could drain the device's battery.
- Two main problems of energy inefficient application are: Developers lack knowledge on how to measure, profile, and optimize energy efficiency, and they lack tools to help them in these tasks.

Software Development Life Cycle

Phases of Software Development Life Cycle



Different SDLC Methodologies

- Waterfall
- Agile
  - Adaptive Software Development(ASD)
  - Agile Software Process (ASP)
  - Dynamic System Development Method (DSDM)
  - Extreme Programming (XP)
  - Feature-Driven Development(FDD)
  - Lean Development
  - Pragmatic Programming
  - Scaled Agile Framework (SAFe)
  - Rational Unified Process (RUP)
  - Systematic Customer Resolution Unraveling Meeting (SCRUM)
- Other Light weight SDLC
  - Whitewater Interactive System Development with Object Models (Wisdom).

“One size fits all” approach is not appropriate

Waterfall	• Quick to accomplish, suitable for small-to-midsize projects. • The project requirements should be precise and cannot be adjusted on the go.
Scrum	• High adaptivity to requirements changes amidst the project management cycle. • Initial timeframes are adjusted far too often and delivering the expected product precisely on time is not a thing happening often.
Lean	• Elimination of superfluous activity, thereby saving time and money. More functionality to be delivered in a shorter period. • Requires excellent document and heavily team dependent with high skill level.
Wisdom	• Improves usability efficiency, user satisfaction and reduced cost and complexity for highest quality interactive systems. • Wisdom is intended for use by small development teams with more user-centered focus.
Feature Driven Development	• Works excellent for large-scale products requiring constant updating and always delivers value. • High reliance on chief programmer. He act as coordinator, lead designer, and mentor.

Being adaptive

- Assess the projects need, strengths, and challenges.
- Determine the ability and willingness of the development staff to switch from traditional methodologies.
- Investigate the various methods and determine which is the best fit with the current project/Organization.
- Factor affecting selection should include people, criticality and priority as main criteria.

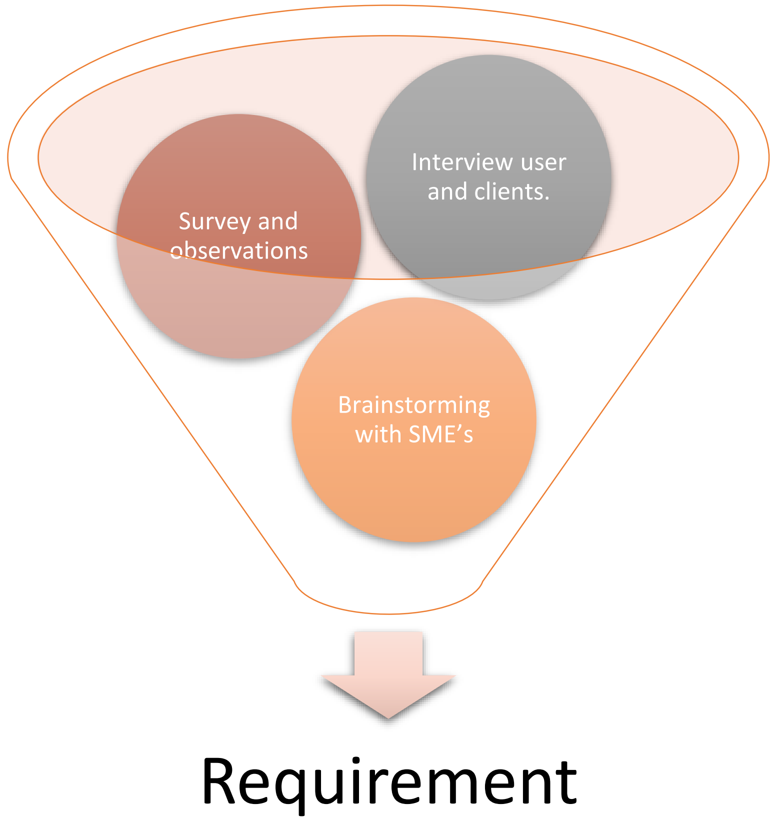
Being adaptive choosing the methodology at the project level rather than organization is something that could save lot of efforts making the all the team adjusting according to the methodologies mandatory requirement.

Few project type examples

- Category 1: An utterly new application from scratch.
- Category 2: Adding new feature to the existing application.
- Category 3: Adding an alternative solution of the existing feature.
- Category 4: Migrating the project to new technology or environment.

Category 3rd and 4th sometime achieve the common goals. A solution to category 3 could be identifies as a new technology while searching for alternative solution or requirement of migrating the project could be the need to identify alternate solution

Requirement gathering and analysis



Important points to remember during requirement analysis

1) For the utterly new application first identify the critical requirement. Then further split it into multiple features(if application is big) to perform the additional detail analysis of priority features or combination of similar features. Analyzing the significant requirement not splitting it into feature could lead to missing some crucial points for some feature or focusing on the avoidable points of some low priority features.
2) Migration requirement most of the time require the backward compatibility since some of the customer are unwilling to change the environment for the execution. Resulting in additional cost so it adds the additional cost of the maintenance. Also not each every features of an application is migratable on one go and required phase release.
3) For Alternative solution, the critical thing to note is the potential impact on the user and user willingness to change.
4) For adding the feature to the application, it is always important to notice the impact on performance.

Not every organization can afford to spend a lot of money or resources only on the requirement gathering. There are a lot of ways where the cost-cutting and time-saving are done in this part of software development.

✓Skipping this phase thinking it doesn’t add any real value.

✓Single person gathering requirement with no validations.

✓Talking to just one user.

✓Talking to a large number of users.

✓Only gathering all requirement before any development could begin.

And the list continues. Sometimes the requirement gathering team just forget to identify “**must-have**” and “**nice to have**” elements of the project. Identifying which feature should be the main focus and which could be added as increment later results not only in saving the development time and cost but also getting early feedback of the customer as well as developing a customer base.

Not only identify potential customer but also..

Development time

- Urgency of the product/solution.
- Business impact on delay's.

Development cost

- Check if any good solution is available in market or internally within organization.
- Potential need of the solution.

Execution Efficiency

- Potential hardware platform and its performance
- Possibility of extension/upgrade of hardware if required to make application more compatible.

Energy Efficiency

- Current energy usage vs Battery life.
- Is user most concerned about the energy saving?

A Story – Background

- An application which was developed in 90's on a mainframe platform to correct and update the data and making sure it is compliant and error-free. The mainframe was chosen for 2 reason Speed and security. The application almost consists of code in mainframe assembler language
- Data in the database was a large set of delimited flat record with delimiter as the indicator of the begin of a specific category of data and was used to generate the report.
- Over the time architecture grew and currently have around 5000 module with more than a million lines of code making an application more slower.
- Since data is updated on daily basis, chances of an addition of incorrect data in some segment of the record also became the problem. Sometimes the data correction requirement was more complex and require a to identify a segment on the specific condition and apply delete or update.
- To achieve that application did had place holder program executed in between the flow of the batch run and either had a special requirement or dummy program (Standard entry and exit).
- Traversing flat record and first identifying the correct segment, then identifying if criteria is met, and the applying update usually resulted in 500 – 1000 lines of code.
- And since it uses to be a temporary requirement, we also need to replace the code the next day with the dummy code after the correction is done in batch.
- Data correction is always a high priority and needs to be corrected as soon as possible.
- Seeing to the situation a analysis was done to automate it and reduce time to address data correction of such complex type.

Analysis factors

- ✓ Possible way to achieve the automation.
- ✓ Approximate estimation of addition of code lines and decision making code.
- ✓ Approximate cost estimation to add the code base and possible maintenance cost.
- ✓ Frequency of current requirement (predictive analysis) and future possibilities.
- ✓ Urgency of the solution and approximate cost saving by reducing temporary development and deployment cost.
- ✓ The customer impact on delay of the data correction.
- ✓ Performance impact with the extra execution time for single record, because now code have to go through a additional decision making code.
- ✓ Any hardware upgrade or addition of any new type of environment required to apply best solution.

Factor and calculation lead decision to postpone the solution

- We identified the best possible way is to create a user interface application that generate the record in a specific format for the different requirement and further that record is traversed by some of specific module and apply the necessary updates. During research we found out 7 category of special requirement and would require a 7 module to be added .
- Without this feature a requirement could take around **15 developer day** for end to end implementation (development, testing and deploying), While with this feature process was still taking **3 days (testing and adding requirement record to production file)**. So **saving was 12 developer day per requirement**. And if **we get 2 requirement in 3 month means we save 96 developer day in a year**.
- A rough estimation was done of 270 days for implementation for the 7 assembler module and user interface application. Which is means if we continuously use the application we would require 3 years to reach break even point.
- Let’s consider in future a additional field is added to a segment of the record. There might be the need to update this application to make it compatible. Means addition of more developer effort and could delay reaching break even point.
- Hence it was decided to skip the requirement since it did not add any value. But could affect the performance, energy of the application.

- It is necessary to make sure every detail which could affect the future process is appropriately documented and communicated further. It includes
  - Any assumption made which can be validated in future with the real data and adjusted accordingly.
  - Even if a decision was made to not develop the software solution or add the feature which can be used in future as reference if needed to research again the possibility. Many teams consider this step as extra effort. Documenting it can reduce the repetitive effort by the future team.
- Proper tech specification conversation is a crucial step by avoiding any misinterpretation.

Design

For the utterly new application design is a critical phase to make the decision of environment and potential coding language as it will impact the future of the product.

Migration requirement is more critical as identifying the compatible environment where the effort and adverse effects of migration on some functionality could be minimized. For eg, migrating code outside mainframe to open source can somehow lead to compromise the speed and security of the application.

For an alternative solution, it is essential to identify the drawback or flaws in the current solution and how the new solution could overcome it without impacting the existing functionality. Sometimes even this solution can be dropped or postponed if any negative impact is identified in this phase since it could affect the existing user and impact the business.

For adding the feature to the application, the compatibility or possibility to code in the existing environment or modification impact on another feature if required.

Factors affecting decision taken in design phase

- Coding language choice is highly dependent on the area of the organization, the organization workforce skills and financial availability or budget of the project.
- When a single person is making the decision, he might choose the one solution which he is aware of which might be not the best solution.
- Too many views in the design phase delay the kick off of the development due to the different point of opinion and overthinking.
- Not spending much time in design might some time result in development failure as some important blocker which could have been identified at the design phase and lead to think for the alternative solution.
- Spending too much time and always changing the approach on identifying a potential blocker also could affect the timelines. Being adaptive here and identifying the minimum data to start the implementation is also a some time right approach.

Choice of coding language

- Each coding language uses different compilers and it depends a lot on compiler accuracy to convert the code to machine code adding the factors that affects the performance, energy consumption of the application. Knowing the generated byte code of a specific function could help in compensating that factor. Every coding language have its own benefits and drawbacks.
- Also choice could be affected by the availability of the developer and market cost. Sometime limited availability could be compensate by immigrating specific skill people which increase the cost. As alternative instead of immigrating people, training the existing workforce on specific skill reduce the cost to some extend but impact on the time.

Usage of third party libraries.

- Too much usage of third-party libraries might lose the control over application performance and energy consumption. On the other hand, coding everything could delay the implementation and adding extra cost for testing and maintaining the large owned code base.
- Also, choice of correct third-party libraries could be decided during the actual development, but the decision to use it is necessary to be made impacting the future process.

Cloud offloading [1]

- Cloud computation offloading is a process that sends heavy computation to resourceful servers on the cloud and then receiving the results from them.
- Offloading brings many potential benefits, such as energy saving, performance improvement, reliability improvement, ease for the software developers, better exploitation of contextual information. The main advantage of cloud computing is its elastic execution that resources can be obtained on demand, it is an enabler for computation offloading.
- It is challenging to run very complex applications on mobile devices because of the strict constraints on their resources such as memory capacity, network bandwidth, CPU speed, and battery power. Servers are resource-rich while the mobile device is resource-limited.
- Cloud offloading has the potential to save execution time and energy consumption but the savings from offloading the computation need to exceed the time and energy cost due to the additional communication between mobile devices and cloud.
- The network bandwidth has a significant impact on performance indicator. Processing on the cloud requires additional data communication, which may increase the time and the battery consumed by communication.

If execution time on the mobile device > (execution time on the server + (exchanged data/network bandwidth)) it is beneficial for the performance efficiency.

If (power for computing * execution time on the mobile device) is greater than ((execution time on the server * power while being idle + (power for sending and receiving data *(exchanged data/network bandwidth))) it is beneficial for energy efficiency

Some additional research examples

- Li et al. [2] analyzed more than 400 real-world Android apps, and found that an HTTP request is the most energy-consuming operation of the network.
- Chowdhury et al. [3] observed that HTTP/2 is more energy efficient than its predecessor, HTTP/1.1, for networks with higher Round Trip time (RTTs).

Some key points to remember.

- Not every time the first available solution is the best solution.
- For any type of requirement, it is always worth to identify the third party libraries which could be a complete solution or merely reduce the coding efforts.
- If a choice is to be made for software language and platform, research the way which can make the application such flexible and easy to migrate or platform independent in case of future.
- Sometimes having a prototype or Minimum viable product in this phase could lead in early feedback which could help to implement the application in the right way.
- Not clarifying the assumption made in requirement phase and further assuming the requirement might result in the modifying too much of the developed code in future.

Implementation/Maintenance and Testing

Efficiency in this phase depends on some critical factors including the individual efforts to interact with the system, tools, and processes, good compilers availability, the efficiency of an operating system.

Expecting the efficiency from day one is very ideal situation but almost complex to achieve.

Developers are not afraid of hard work, they are afraid of failure, and if you set them up to succeed with definite plans, they will not only work hard but take ownership and increase effectiveness.

For all the category of the project the point to remember remains almost the same.

Some long term solutions

- Reusability - Determining patterns early on and writing modular code will help you to simplify your process and increase reusability. Planning out smaller functions that can complete one task is an easy way for programmers to make code reusable and efficient.
- Documentation - Improving your documentation and keeping everything updated and prioritized will significantly help in your development process and tracking all the assumption to be taken care later.
- Splitting - Splitting entire project into small stages and gradually adding new features at each new stage can help in identifying a impact on application efficiency on addition of any specific feature and help in benchmarking the product effectively
- Refactoring
- Continuous integration.
- Research few beneficial third party libraries.
- Enhancement and technical debts.

Refactoring [4]

- An example of ConcurrentHashMap and Hashtable which implement the Map interface.
- It was observed in one experiment that just updating from Hashtable to ConcurrentHashMap in a Java program can yield a 3.5x energy savings.
- ConcurrentHashMap uses multiple buckets to store data. This avoids read locks and dramatically improves performance over a Hashtable.
- But, the significantly process is not always straightforward.
- 1) Hashtable inherits instance from the dictionary and implements cloneable, while ConcurrentHashMap does not. So hash.clone() raises a compile error.
- 2) Third party libraries often require implementations instead of interfaces. If a method is expecting a Hashtable instead of, say, a Map, a ConcurrentHashMap needs to be converted to Hashtable, decreasing its effectiveness.

Continuous integration

Not every time continuous integration is beneficial. It is more useful when the development is split into multiple features and the number of developers is working separately on a separate feature.

It helps in the following ways

- Quarantining errors
- Validating assumptions
- Analysis and reporting reducing the risk of any missed vulnerability.
- Prevention & reduction of production & staging errors.

Continuous integration is an ongoing process and gets improve over time. Spending initial effort to set up is important and necessary when the development efforts are longer estimated and team wanted to make sure adding some new feature does not affect the previously added/updated features.

The decision to have continuous integration increases the cost and initial time of the development, but help in reducing the cost and time spend in maintenance since many errors could be identified and solved during the development.

Beneficial third party libraries.

- Adding third-party libraries reduce the cost of development and development time if they are useful and cost are low.
- Since it is 3rd party code, the maintenance and support cost also get reduced and development team can focus on "**must have**" features. The decision to include the libraries it to be validated with the facts that the performance degradation and energy efficiency are not too much compromised.
- Also sometime during the maintenance, it becomes hard to predict the actual source of the problem (App or 3rd party).
- Adding too many third party libraries also create problem to manage them and additional effort to make sure that any depreciation in the libraries are appropriately addressed in time and impacted code changes are made.
- Also sometime the security could be compromised with the Risk of malicious 3rd party intent. Also, it is not the right approach to develop everything from scratch if some good solutions are readily available.

Enhancement

- An enhancement is any product change or upgrades that increase software or hardware capabilities beyond original specifications.
- In general, product enhancements include:
  - Additional functionality.
  - Error/bug repair and handling.
  - Greater processing speed.
  - Better cross-platform compatibility.
- Expected challenges for adding a functionality.
  - The changes made could degrade the performance to which users are accustomed.
  - Users who are familiar with how the system works today might not like the changes they are about to encounter (For ex. Changes done for better UX changes)
  - You might unknowingly break or omit functionality that is vital to some stakeholder group.

Technical debts [5]

- Technical debt refers to a concept in software development where extra development work arises, through intentional decision or unintentional side-effect, when code that is easy to implement in short-run is used instead of applying the best overall solution. Technical debt has three main attributes that are similar to financial debt: Repayment, interest and high cost.
- Some time with the scarcity of budget and timeline, a quick solution is used to make the feature available, to avoid loss of any user. Document such technical debt and prioritize them to avoid the future maintenance and any performance or energy degradation.
- In software development, when too much of technical debt has been taken and not repaid in time, the overall quality might decrease under a certain level, where it may not be possible to do any changes to the software without breaking it, due to complexity. In worst cases, this could mean the end of the software.

Type of basic testing

Development

Helps to identify the working of the functionality by manually requesting the functionality and checking the response to validate it is as per requirement.

Unit

Helps to identify the working of a specific function or a module on different conditions.

Its good to have in a big application's to avoid future change breaking the functionality of a function.

Integration/Functional

Helps to identify the working of specific functionality/request and checking the output on different input and exception conditions.

Performance

Identify application response time on the massive user workload by virtual injection of multiple users to request same functionality at the same time.

Penetration

It is an attempt to evaluate the security of an IT infrastructure by safely trying to exploit vulnerabilities.

The weak points of a system are exploited in this process through an authorized simulated attack.

Some decision taken and impact considered

- We invested 15-20 days to implement continuous integration on the product which helped us in identifying many old bugs and potential enhancement due to performance degradation and possible security leak. Since the agent uses a lot of libraries there is an additional effort to maintain it which was identified as negligible in our case.
- The application was refactored to move duplicate code on identifying a lot of duplicate code in the application. With the duplicate code, the codebase increased and required more effort to maintain and track it. So the decision was made at the first level to move the duplicate code which is repeated for 5 or more places to a common module as a part to identify the required effort and impact on cost and performance. It did increase the additional processing to call the function with parameter and return a result, Additional cost and time to identify and refactor the code. But it reduce the memory usage hence energy efficiency, development time for the future request and maintenance and also few routine were even refactored to improve the performance by removing extra unwanted processing.

Deployment

Software deployment process consists of all the activities that make a **software** system available for use.

Some of the activities include release, Installation and activation, Deactivation, Uninstallation, Update, Built-in Update, Version tracking, and Adaptation.

The complexity and variability of software products have fostered the emergence of specialized roles for coordinating and engineering the deployment process. It includes application developers, builds and release engineer, deployment coordinators as well as the system administrator, database administrator, release coordinators.

Deployments that take weeks to complete, slowed down by too much manual intervention and could even delay sometime due to system limitations and some human errors.

Choice of deployment process affects the release time and cost.

Not every product needs a continuous deployment at the first release or even if the product having very infrequent releases. But if the product has persistent updates and release it is always worth to spend some time investigating in continuous deployment to avoid the delay of releases and saving time by reducing the manual efforts.

Few observations and conclusions

- Being adaptive to the change and making the software more flexible is the most crucial decision.
- Sometimes the team/software need to compromise one of the application efficiency. Try to not over compromise one of the factor to achieve the another.
- Not every solution works in every situation. “One size fits all” is the biggest misconception across and need to be avoided as soon as possible during the cycle.
- Try to get as much as early feedback from the user. A minimum viable product is a good choice when the requirement is not clear. “Minimum” should not be treated as a compromise version.

References (https://www.researchgate.net/)

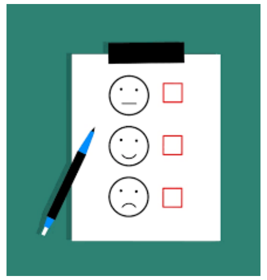
- [1] Huaming Wu and Katinka Wolter. Conference: IEEE International Conference on Communications 2013: IEEE ICC'13-1st International Workshop on Mobile Cloud Networking and Services (MCN) At: Budapest Hungry.
- [2] Chowdhury, S.A., Sapra, V, and Hindle, A. Client-side energy efficiency of HTTP/2 for web and mobile app developers. In *Proceedings of SANER*, 2016, 529–540.
- [3] Li, D., Hao, S., Gui, J., and Halfond, W.G.J. An empirical study of the energy consumption of Android applications. In *Proceedings of ICSME*, 2014, 121–130.
- [4] Pinto, G., Liu, K., Castor, F., and Liu, Y.D. A comprehensive study on the energy efficiency of Java thread-safe collections. In *Proceedings of ICSME*, 2016.
- [5] Jesse Yli-Huumo. The role of technical debt in software development. Thesis for: Doctor of Science.
- [6] Gustavo Pinto, Fernando Castor. Communications of the ACM, December 2017, Vol. 60 No. 12, Pages 68-75 10.1145/3154384

We want your feedback!

- Please submit your feedback online at ….
  - ➢http://conferences.gse.org.uk/2018/feedback/MM

- Paper feedback forms are also available from the Chair person

- This session is MM



Many Thanks!
THANKS