IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 6, Issue 11, November 2017

Collaborative Requirements Elicitation and Development Methodology for Migrating from SaaP To SaaS

Mr. Mahesh Nelapati[1], Mr. Simegnew Asemie[2], Mr. Tefera Adugnaw Lulie[3]

Dept., of Information Technology School of Computing and Informatics, Mizan-Tepi University, Tepi Campus, Ethiopia[1-3]

Abstract: In present days to determine trends in Information systems, organizations expansively used Service Oriented Architecture technology. Tradesman of Information system products want to advantage by immigrating to central based internet. Despite when immigrating legacy and traditional information system from software as a product model to the software as a service model the information system model changes. This paper explicates the obligatory modifications in the requirements elicitation process and producing a methodology for a strong immigration to software as a service. This paper elaborates by deliberating collaborative requirements information system development methodology. This methodology analyzes initial stage grade that must be deliberating in software as a service and we can reduce risk factors. Additionally this methodology discusses the new advantages in requirements elicitation that are deriving in to central based internet.

Keywords: Information systems, Service oriented technology, Central based internet, Software as a service (SaaS), Collaborative requirements.

1. INTRODUCTION

Thinks about demonstrate that 37% of the IT associations consider utilizing SaaS as extremely important. Majority of IT masters this point is normal significance some companies not interested since security issues, execution and accessibility e.t.c. Another review calls attention to that contracting a product as opposed to obtaining yields in a sparing of 45% of the client's costs in a three year time traverse [12]. Long, long back in the ancient past days - before the late '90s - programming was sold as an independent item that you sourced, introduced, arranged, secured, directed, refreshed and after that in the long run utilized. Contingent upon the application, you could invest a considerable measure of energy and cash sending and designing a bundle before at long last finding the opportunity to really run it. At that point as new upgrades/elements were craved and made accessible, more tweaking was required - and likely more equipment was additionally required.

The original of SaaS, additionally called "on request" conveyance, endeavored to improve some of these cerebral pains. It was - and still is - a product conveyance display where the application is facilitated remotely and clients get to it through the web. The product supplier takes care of the application's operation, upkeep, security and support - all the confounded, tedious undertakings. The end client benefits by going out on a limb when making an innovation venture, while additionally getting a charge out of a diminishment in the aggregate cost of proprietorship. Basically, you pay an expense, utilize your program to sign in, do what you have to do, then log off. Another person needs to stress over the various stuff.

A PaaS supplier has the equipment and programming all alone framework or an employed foundation, for example, AWS. Subsequently, PaaS soothes its clients from introducing in-house equipment and programming expected to create or run another application. PaaS enables the engineer to concentrate on the application by covering all foundation and middleware related administration perspectives.

IaaS stages have versatile parts and can be balanced as required. They are consequently perfect for variable, test and impermanent workloads. The model functions as takes after: an outsider has clients' product, stockpiling, equipment, and other foundation segments. Client applications are likewise facilitated by IaaS suppliers, and they additionally oversee errands, for example, reinforcement, framework support, and versatility arranges. Different elements of IaaS include mechanized organization errands, desktop virtualization, deft scaling, and arrangement based administrations.
IaaS suppliers some of the time charge clients for the measure of virtual machine space they utilize, however ordinarily IaaS clients pay per-use by the month, week or hour. This model removes the cost of setting up in-house programming and equipment. It is prescribed, in any case, that clients check their IaaS administration to guarantee they are not being charged for unjustifiable administrations. These three cloud services communication process shown in below Figure-1.

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 6, Issue 11, November 2017

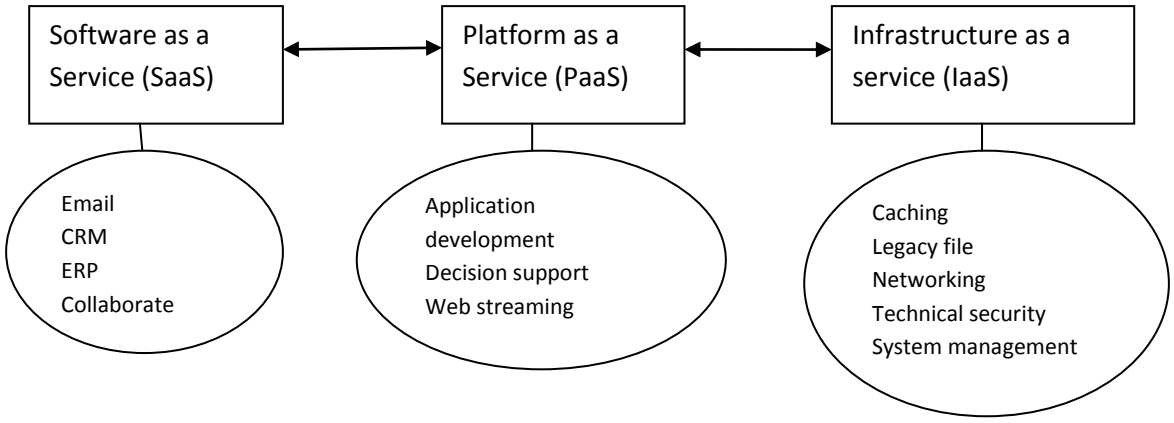

Figure-1: Cloud computing service model

2. BACK GROUND

For a long time, programming has been delivered in a supplyside situated way [12]. A product seller places eff ort into the necessities elicitation for a specific issue, creates and tests the product and discharges the final item to the market. The client or the product merchant's bolster group introduces a duplicate of the product item at the client's framework subsequent to acquiring a permit. While minor programming updates are generally led by means of an Internet interface and incorporated into the one-time value, significant redesigns frequently require purchasing another product item [3]. The SaaS show, in correlation, is the pattern in programming building of the 21st century that difficulties this customary model [1]. The client of a SaaS-based programming buys a use ideal for a specific time traverse. Consequently, the seller awards access to the online administration.

Since its first says in research in the 2000s, SaaS has increased increasingly consideration both from scientific and creation perspectives. While diff erent approaches –, for example, iterative and incremental improvement forms and secluded programming items – have been set up to address the issues of creating and sending more mind boggling programming items, the SaaS model is a radical move of the methods by which programming is built. Giving Software as a Service as opposed to an item, at a first look, is a way of circulation approach aff ecting business issues like time to market, client association and discharge cycles [8]. The administration introduction of programming, be that as it may, likewise accompanies real worldview changes with respect to the product improvement. The SaaS demonstrate uses benefits as the simple variable for arranging the many-sided quality of programming [9]. These administrations are made accessible by specialist co-ops that surfaced with the administration foundation and the usage and give the interface portrayal to access over the Internet (electronic). With a specific end goal to distribute and find incorporation prepared administrations, a typical administration registry is required (see Figure-2). Administrations itself are made out of different administrations recursively [3].

3. RELATED WORK

Since these seminal works, research has made a lot of progress. In their study [6] Kumar and Sangwan present traditional software engineering process models and main concepts (e.g. Iterative development).
They continue collecting aspects which make the development of web-based applications different from traditional software. According to the authors the main aspect is the continuity of the process that also requires a systematic, repeatable and iterative process. Together with lists of attributes and characteristics of web-based applications they provide a very general adaption of a traditional software engineering process model towards a model which is suitable for web-based applications. However, the authors fail to present a detailed process as a result that can be used for developing such applications. Nogueira da Silva and Lucr´edio [11] have also conducted an extensive literature review.

They found out that the research interest has increased over the last years. They identify the main challenge for cloud-based software engineering to be the lack of standardization. E.g. choosing a PaaS provider may result in platform lock-ins where customers cannot easily switch to another service provider. Besides a grouping and the presentation of challenges for SaaS developers the authors provide definitions of the terms SaaS and SOA. They conclude that a research gap exists regarding the formalization of a complete reengineering process in terms of reconstructing the software for a new platform. Balian and Kumar [2] group and review studies in the field of SaaS development. They list technical non-functional requirements, legal concerns and other issues from the data management. The authors then categorize these topics and identify the new stakeholder cloud service provider. As a result, they propose an addition to the Capability Maturity Model Integration (CMMI) reference model that includes a checklist for the new stakeholders. Research has provided detailed descriptions of the SaaP model and the fundamentals behind the SaaS model. Recent

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 6, Issue 11, November 2017

work also exists which covers the transformation of a service oriented system into a cloud-based software that follows the SaaS model [4] [5] [7] [10] [13].

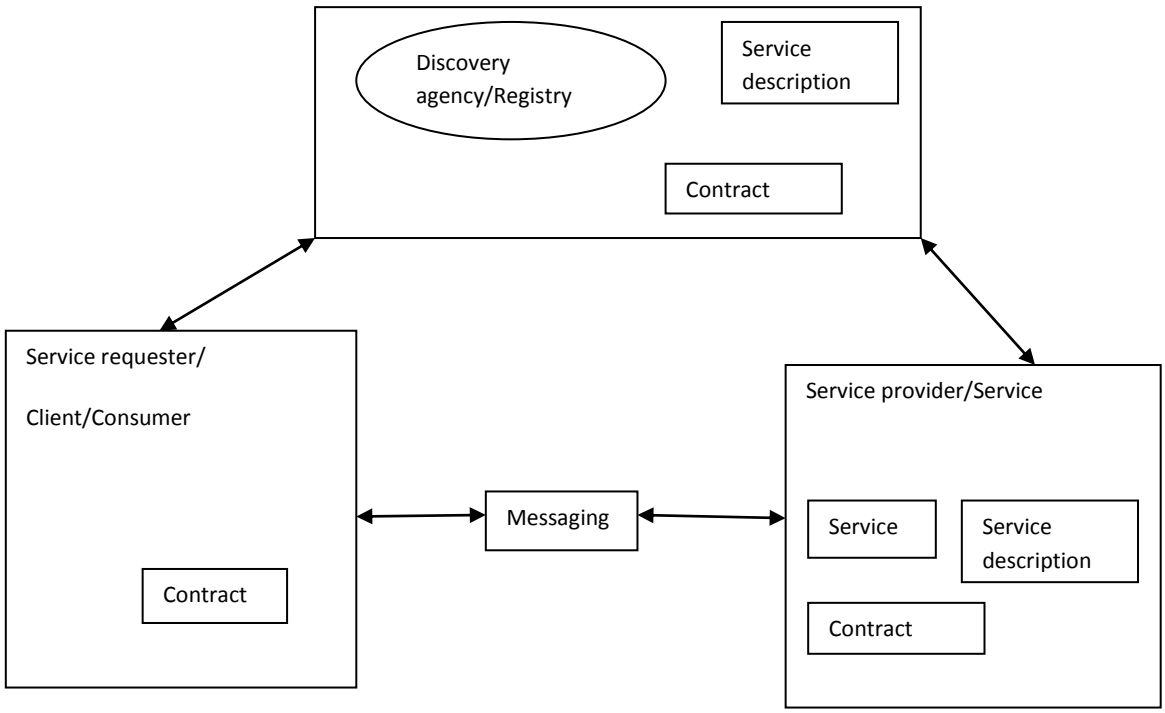

Figure-2: Service oriented architecture model

4. REQUIREMENTS ELICITATION AND DEVELOPMENT APPROACH

Precise Methodology:
Start and Planning:
Prerequisites elicitation joins components of critical thinking, elaboration, arrangement, and detail. So as to energize a communitarian, group arranged way to deal with prerequisites gathering, partners cooperate to distinguish the issue, propose components of the arrangement, arrange diverse methodologies and determine a preparatory arrangement of arrangement necessities. Community oriented prerequisites elicitation:

1. Gatherings are led and gone to by both programming engineers and different partners.
2. A motivation is recommended that is sufficiently formal to cover terrifically imperative indicates yet sufficiently casual support the free stream of thoughts.
3. For consumer loyalty need to take after Normal necessities, expected prerequisites and leaving prerequisites. There are five general strides in "hypothesizing," in spite of the fact that "means" is to some degree a misnomer, as each progression might be updated a few times amid the start and arranging stage.

Step-1: Product start includes setting the item's main goal and targets, understanding imperatives, building up the item association, recognizing and illustrating prerequisites, making beginning size and extension evaluates, and distinguishing key item hazards. Since speed is normally a noteworthy thought in utilizing ASD, a great part of the item start information ought to be assembled in a preparatory JAD session. Start can be finished in a concentrated two-to five-day exertion for a little to medium-sized item or take half a month for bigger items. Amid the JAD sessions, prerequisites are assembled in enough detail to distinguish includes and set up a skeletal protest, information, or other building model.
Step-2: Next, the time-box for the whole item is set up in light of the degree, include set necessities, assessments, and asset accessibility that outcome from item start work. Theorizing doesn't relinquish assessing; it just means tolerating that appraisals are questionable.
Step-3: The third step is to settle on the quantity of cycles and relegate a period box to everyone. For a little to medium-sized application, emphases more often than not shift from four to two months. A few items work best with two-week cycles, while others may require over two months (in spite of the fact that this is uncommon). The general item estimate and the level of vulnerability are two variables that decide singular cycle lengths.

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 6, Issue 11, November 2017

Step-4: After building up the quantity of cycles and a timetable for each, the colleagues build up a topic or target for each of the emphases. Similarly as it is vital to build up a general item objective, every cycle ought to have its own particular topic (this is like the Sprint Goal in Scrum). Testing is a continuous, essential piece of highlight improvement—not an action attached on toward the end.
Step-5: Developers and clients allocate elements to every emphasis. The most vital standard for highlight task is that each emphasis must convey an unmistakable, substantial arrangement of elements to the client. In the task procedure, clients settle on highlight prioritization, utilizing highlight appraisals, dangers, and reliance data provided by the advancement group. A spreadsheet is a compelling apparatus for highlight based emphasis arranging.

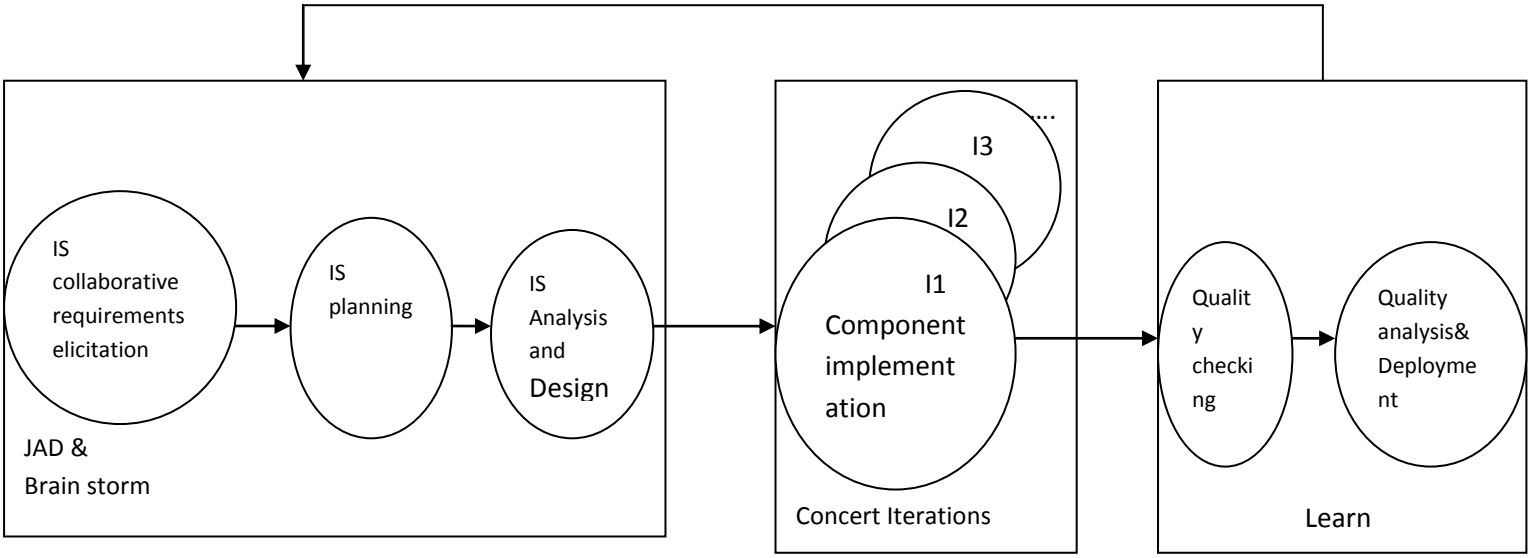

Figure-3: Collaborative requirements elicitation and development methodology

Encounter has demonstrated that this sort of arranging—done as a group instead of by the item director—gives preferred comprehension of the item over a conventional undertaking based approach. Include based arranging mirrors the uniqueness of every item. The overall process methodology shown in the above Figure-3.

Component Development:
While the specialized group conveys working programming, item administrators encourage joint effort and simultaneous advancement exercises. For items including dispersed groups, changing collusion accomplices, and expansive based information, how individuals connect and how they oversee interdependencies are key issues. For littler items in which colleagues work in physical closeness, cooperation can comprise of casual corridor visits and whiteboard writing. Bigger items, be that as it may, require extra practices, cooperation apparatuses, and item director communication, cooperation, a demonstration of shared creation, is cultivated by trust and regard. Shared creation includes the advancement group, clients, outside experts, and sellers. Groups must team up on specialized issues, business prerequisites, and quick basic leadership.

Quality Checking and Deployment:
Learning turns out to be progressively troublesome in conditions in which the "hit the nail on the head the first run through" mantra overwhelms and improvement advances in a straight, waterfall form. On the off chance that individuals are ceaselessly constrained to hit the nail on the head, they won't analyze and learn. In waterfall improvement, each stage finish debilitates backtracking on the grounds that there shouldn't be any oversights. Gaining from missteps and experimentation requires that colleagues share somewhat finished code and curios ahead of schedule, with a specific end goal to discover botches, gain from them, and decrease the aggregate sum of revise by discovering little issues before they turn out to be vast ones. Groups must figure out how to separate between trashy function and half-done work. There are four general classifications of things to find out about toward the finish of every improvement emphasis:

1. Result quality from the client's point of view.
2. Result quality from a specialized point of view.
3. The working of the conveyance group and the practices colleagues are using.
4. The item's status.

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 6, Issue 11, November 2017

Contrast between SaaP and SaaS:

SaaP	SaaS
Time setting aside process	Reduced opportunity to profit/Rapid prototyping
Higher cost of entry	Lower cost of entry
Complicated to foresee acceptance	Can effectively anticipate acknowledgment
Vendor Not in charge of upgrade	Vendor is in charge of redesign
Vendor Not capable uptime and security	Vendor is in charge of uptime ,further more security

Programming process change:

Change property	Improvement rate
Development cost decrease	75%
Rework cost decrease	96%
Project risk reduce	96%

5. DISCUSSION AND PRACTICES

The fundamental objective of this paper was to plot the deference between the necessities designing procedure of a customary programming item and the procedure of a product benefit and to give a methodical way to deal with relocating from one to the next. Taking after the displayed approach lessens the hazard for forgetting essential changes in the prerequisites building process. For the individuals who consider relocating a product item yet have not chosen yet, the approach defines the extent of changes which would be natural for an arranged movement. Accordingly this current paper's approach offers benefits that can't be found in writing starting today.

6. LIMITATION & CONCLUSION

This approach covers the prerequisites designing procedure, which is just a single some portion of others in the entire programming advancement prepares. The relocation of programming items to the cloud can in any case bomb because of different ramifications of such a procedure movement. And with this methodology can reduce development cost, rework cost and risk factors.

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al. A view of cloud computing. Communications of the ACM, 53(4):50–58, 2010.
[2] N. Baliyan and S. Kumar. Towards software engineering paradigm for software as a service. In Contemporary Computing (IC3), 2014 Seventh International Conference on, pages 329–333. IEEE, 2014.
[3] K. Bennett, P. Layzell, D. Budgen, P. Brereton, L. Macaulay, and M. Munro. Service-based software: The future for flexible software. In Software Engineering Conference, 2000. APSEC 2000. Proceedings. Seventh Asia-Pacific, pages 214–221. IEEE, 2000.
[4] M. A. Chauhan and M. A. Babar. Migrating service-oriented system to cloud computing: An experience report. In Cloud Computing (CLOUD), 2011 IEEE International Conference on, pages 404–411. IEEE, 2011.
[5] M. A. Chauhan and M. A. Babar. Towards process support for migrating applications to cloud computing. In Cloud and Service Computing (CSC), 2012 International Conference on, pages 80–87. IEEE, 2012.
[6] S. Kumar and S. Sangwan. Adapting the software engineering process to web engineering process. International Journal of Computing and Business Research, 2(1), 2011.
[7] G. Lewis, E. Morris, and D. Smith. Service-oriented migration and reuse technique (smart). In Software Technology and Engineering Practice, 2005. 13th IEEE International Workshop on, pages 222–229. IEEE, 2005.
[8] E. R. Olsen. Transitioning to software as a service: Realigning software engineering practices with the new business model. In Service Operations and Logistics, and Informatics, 2006. SOLI'06. IEEE International Conference on, pages 266–271. IEEE, 2006.
[9] M. P. Papazoglou. Service-oriented computing: Concepts, characteristics and directions. In Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on, pages 3–12. IEEE, 2003.
[10] E. Saleh. Migrating traditional web applications into multi-tenant saas. Proc. 6th Ph. D. Retreat of the HPI Research School Serviceoriented Syst. Eng, pages 145–155, 2013.
[11] E. A. N. d. Silva and D. Lucr´edio. Software engineering for the cloud: A research roadmap. In Software Engineering (SBES), 2012 26th Brazilian Symposium on, pages 71–80. IEEE, 2012.
[12] A. Tariq, S. A. Khan, and S. Iftikhar. Requirements engineering process for software-as-a-service (saas) cloud environment. In Emerging Technologies (ICET), 2014 International Conference on, pages 13–18. IEEE, 2014.
[13] X. Zhang, B. Shen, X. Tang, and W. Chen. From isolated tenancy hosted application to multi-tenancy: Toward a systematic migration method for web application. In Software Engineering and Service Sciences (ICSESS), 2010 IEEE International Conference on, pages 209–212. IEEE, 2010.