American Journal of Information Science and Technology
2018; 2(2): 50-56
http://www.sciencepublishinggroup.com/j/ajist
doi: 10.11648/j.ajist.20180202.14

Transitions in System Analysis and Design Methodology

Nwakanma Ifeanyi Cosmas, Adu Folashade Christiana, Okafor Obinna Jeremiah, Akuta Cletus Ikechukwu

Information Management Technology, Federal University of Technology, Owerri, Imo State, Nigeria

Email address:

fraircos@yahoo.com (N. I. Cosmas), xtiana1202@gmail.com (A. F. Christiana), okaforj@yahoo.com (O. O. Jeremiah), plusforesight@yahoo.com (A. C. Ikechukwu)

To cite this article:

Nwakanma Ifeanyi Cosmas, Adu Folashade Christiana, Okafor Obinna Jeremiah, Akuta Cletus Ikechukwu. Transitions in System Analysis and Design Methodology. *American Journal of Information Science and Technology*. Vol. 2, No. 2, 2018, pp. 50-56.
doi: 10.11648/j.ajist.20180202.14

Received: April 6, 2018; **Accepted**: May 10, 2018; **Published**: July 4, 2018

Abstract: Systems Analysis and Design is an exciting endeavour as well as an active field in which analysts continually learn new techniques and approaches to develop systems more effectively and efficiently. Any organization that wants to have a long-lasting impact on its target market, must be ready to invest its resources in planning and research, to ascertain whether a new project is viable, partially viable or impracticable. This will either show the survival tendencies of the organization as it relates to the project or its weaknesses in handling the project. Every system development inadvertently follows four phases, which are: planning, analysis, design, and implementation. All complex systems can be decomposed into a nested hierarchy of subsystems because the different facets of every individual organization are either a system, part of a system or a subsystem. For an embodiment of various singular interconnected parts to be considered a system, it must have followed through with a methodology or an approach. This paper utilizes expository methodology and drives towards giving a concise overview of the various approaches to be adopted while developing a system. It begs to give more insight to the current methodologies in systems development, the emerging approaches and the pros and cons. The authors investigate from inception to the current methodologies, knowing full well that many occurrences in life happens in correspondence to dispensations, times and seasons; just like winter and summer, the authors understudy the various dispensations to pin the prevalent methodologies in certain time spaces, the advances or improvement as well as the advantages and disadvantages they have over others as time progresses. In the process, old systems methodologies are improved to serve a larger target and the amount of work needed per time reduces as new system methodology are developed over time.

Keywords: System Analysis, Design, Methodology, Complex System

1. Introduction

A system is composed of interrelated subsystems, each of the latter being, in turn, hierarchic in structure until we reach some lowest level of elementary subsystem [13]. “Every system development inadvertently follows four phases, which are: planning, analysis, design, and implementation” [1]. “Systems development life cycle (SDLC) can be the oldest formalized methodology framework for building information systems” [12]. “All complex systems can be decomposed into a nested hierarchy of subsystems” [16]. Critically however, not all these subsystems are of equal importance (i.e., centrality). Some subsystems are “core” to system performance, whereas others are only “peripheral” [14]. System development follows a process line. System development process is the process of dividing system development work into distinct phases to improve design, product management, and project management.

2. Phases of Sdlc

We have different phases in system development life cycle; and they include the planning; analysis; design and implementation.

2.1. Planning

The planning phase is the fundamental process of understanding why an information system should be built and

determining how the project team will go about building it. It has the project initiation sub phase, and here the system's business value to the organization is identified: Questions like:

How will it lower costs or increase revenues?

2.2. Analysis

The analysis stage is very critical. The analysis phase answers the questions:

1. who will use the system?
2. what the system will do?
3. where and when it will be used?

During this phase, the project team investigates any current system (s), identifies opportunities for improvement, and develops a concept for the new system. In this phase an analysis strategy is developed to guide the project team's efforts. Such a strategy usually includes an analysis of the current system (called the as-is system) and its problems and then ways to design a new system (called the to-be system). After developing the analysis strategy, requirements gathering kicks off (e.g., through interviews or questionnaires).

2.3. Design

The design phase decides how the system will operate, in terms of the hardware, software, and network infrastructure; the user interface, forms, and reports; and the specific programs, databases, and files that will be needed. Although most of the strategic decisions about the system were made in the development of the system concept during the analysis phase, the steps in the design phase determine exactly how the system will operate.

2.4. Implementation

The final phase in the SDLC is the implementation phase, during which the system is built (or purchased, in the case of a packaged software design). This is the phase that usually gets the most attention, because for most systems it is the longest and most expensive single part of the development process.

2.5. Systems Development Methodologies

A methodology is a formalized approach to implementing the SDLC (i.e., it is a list of steps and deliverables) [1]. There are several system development methodologies, and each one is unique, based on the order and focus it places on each SDLC phase. Some methodologies are formal standards used by government agencies, whereas others have been developed by consulting firms to sell to clients. Many organizations have internal methodologies that have been honed over the years, and they explain exactly how each phase of the SDLC is to be performed in that company. Historically, system development approaches have been progressing and continually new methodologies emerge yearly following a sequence is years.

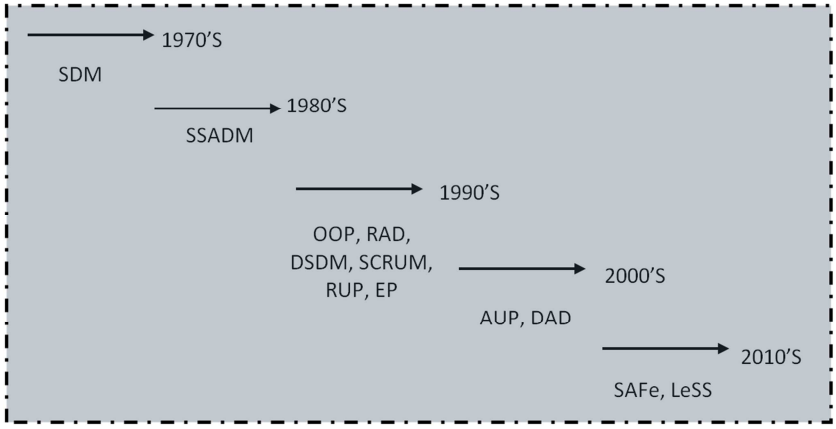

Figure 1. *Dispensation of system development approaches.*

2.5.1. SDM

Structured programming since 1969 as the name suggests deals with system structure. To fully appreciate it, however, one must understand the full extent of the problem addressed by structured programming.

Structured programming is a method of writing a computer program that uses

(1) top-down analysis for problem solving,
(2) modularization for program structure and organization, and
(3) structured code for the individual modules.

Top-down analysis: A program is written to tell a computer what to do. This "job" is more formally called the problem. However, before you can tell the computer what to do, you must "solve" the problem yourself. In other words, you must state every step necessary to accomplish the job. This activity on your part is called problem solving or problem analysis.

Top-down analysis is a method of problem solving. It tells you how to start and guides you through the entire process. The essential idea is to subdivide a large problem into several smaller tasks or parts. Top-down analysis, therefore, simplifies or reduces the complexity of the process of problem solving.

Modular programming: Programs generally require many instructions for the computer. Modular programming is a method of organizing these instructions. Large programs are broken down into separate, smaller sections called modules, subroutines, or subprograms. Each module has a specific job to do and is relatively easy to write.

Structured coding: If programs are broken down into modules, into what are modules subdivided? Obviously, each consists of a set of instructions to the computer. But are these instructions organized in any special way? That is, are they grouped and executed in any clearly definable patterns? In structured programming they arc. They are organized within various control structures. A control structure represents a unique pattern.

2.5.2. SDM2

This is a beta version of SDM also called Cap Gemini SDM, originally from PANDATA, the first English translation was published in 1974 [20]. SDM stands for System Development Methodology. The method is a waterfall model divided in seven phases that have a clear start and end. Each phase delivers (sub) products, called milestones. It was used extensively in the Netherlands for ICT projects in the 1980s and 1990s. PANDATA was purchased by the Capgemini group in the 1980s, and the last version of SDM to be published in English was SDM2 (6th edition) in 1991 by CAP GEMINI PUBLISHING BV. The method was regularly taught and distributed among Capgemini consultants and customers, until the waterfall method slowly went out of fashion in the wake of more iterative extreme programming methods such as Rapid application development, Rational Unified Process (RUP) and Agile software development.

2.5.3. SSADM

Structured systems analysis and design method from 1980 onwards is a well-defined approach. It's not new. The term structured is borrowed from Structured Programming (Randall, 1981). The word structured generally imposes a structure or a disciplined approach on the design of the system. SSADM is in fact a modified form of SDLC. Hence, we can also call SSADM as SDLC using structured techniques.

It consists of:

1. System Survey
2. Structured Analysis
3. Structured Design
4. Hardware Study
5. Implementation and
6. Maintenance

2.5.4. OOP

Object-oriented programming/Methodology (OOP) developed in the early 1960s and became a dominant programming approach during the mid-1990s as an approach to of Building Systems takes the objects as the basis. For this, first the system to be developed is observed and analysed and the requirements are defined as in any other method of system development. Once this is done, the objects in the required system are identified. For example, in case of a Banking System, a customer is an object, a chequebook is an object, and even an account is an object [9]. The basic steps of system designing using Object Oriented Methodology can be listed as:

1. System Analysis
2. System Design
3. Object Design
4. Implementation

2.5.5. RAD

Rapid application development (RAD), has been on since 1991. The work of Boehm and Gilb paved the way for the formulation of the methodology called Rapid Iterative Production Prototyping (RIPP) at DuPont in the mid-to-late 1980s. James Martin then extended the work done at DuPont and elsewhere into a larger, more formalized process, which has become known as Rapid Application Development (RAD). RAD compresses the step-by-step development of conventional methods into an iterative process. The RAD approach thus includes developing and refining the data models, process models, and prototype in parallel using an iterative process.

The challenges facing software development organizations can be summarized as more, better, and faster. The RAD development path attacks these challenges head-on by providing a means for developing systems faster, while reducing cost and increasing quality. Fundamentals of the RAD methodology thus include:

1. Combining the best available techniques and specifying the sequence of tasks that will make those techniques most effective
2. using evolutionary prototypes that are eventually transformed into the final product.
3. Using workshops, instead of interviews, to gather requirements and review design.
4. Selecting a set of CASE tools to support modelling, prototyping, and code re-usability, as well as automating many of the combinations of techniques.
5. Implementing time boxed development that allows development teams to quickly build the core of the system and implement refinements in subsequent releases
6. Providing guidelines for success and describing pitfalls to avoid [5].

2.5.6. DSDM

Dynamic systems development method (DSDM), since 1994, the DSDM Project Framework embraces the project delivery values and fully aligns with the product development philosophy inherent in Scrum. The DSDM Philosophy is that any project must be aligned to clearly defined strategic goals and focus upon early delivery of real benefits to the business. This is best achieved when all stakeholders understand the business objectives, are empowered to an appropriate level, and collaborate to deliver the right solution. This solution will be delivered in the

agreed timescale, according to the priorities driven by the business. As the project progresses, the stakeholders must accept that change is inevitable as the understanding of the solution deepens.

The eight DSDM principles underpin the Project Framework and support the Philosophy. They bring the Values to life by guiding the attitude that must be taken and the mind-set that must be adopted to deliver consistently whilst still remaining flexible. Compromising any principle undermines the basic philosophy and introduces risk to the successful outcome of the project. The eight Principles are:

1. Focus on the business need
2. Deliver on time
3. Collaborate
4. Never compromise quality
5. Build incrementally from firm foundations
6. Develop iteratively
7. Communicate continuously and clearly

2.5.7. Scrum

Since 1995 is the most widely practiced Agile process, has been successfully used in software development for the last 20 years. While Scrum has been mostly practiced in a commercial software environment, the methodology has been successfully applied to education, manufacturing and an array of other industries.

Scrum uses the Divide and Conquer rule. Scrum divides complex work into simple pieces, large organizations into small teams and far-reaching projects into a series of short time horizons called sprints.

When complex work is divided into simple pieces it is easier to map out what needs to be done. With a clear roadmap the team can start working immediately, know what items need to be worked on together and understand when the job has been completed.

Scrum begins with the product vision. The product owner translates the vision into the product backlog. Once the product backlog has been established, the team can start sprinting. To start a sprint, the team must first conduct sprint planning. Sprint planning should be limited to no more than two hours for every week of sprint. The idea behind sprint planning is to have one comprehensive meeting that maps out what needs to be done by the end of the Sprint. An item in the backlog is ready if it is independent, actionable, has been assigned a point value and has a clear definition of the criteria that means it is done. This in turn is referred to as the definition of done which ensures everyone knows exactly what is expected of an item when it is delivered. The team creates a sprint goal once the sprint backlog has been created. This goal should articulate the high-level purpose of the items in the sprint backlog. In simpler terms, the goal provides context for why the team is working on the selected backlog items. A goal may be as simple as reaching a certain level of functionality. "By the end of this sprint the team will demonstrate how the program can save e-mail addresses automatically." After sprint planning the team gets to work and meets every day for the daily scrum. During the daily scrum, each team member answers three questions:

1. What did I do yesterday that help the Team meet the Sprint Goal?
2. What will I do today to help the Team meet the Sprint Goal?
3. Do I see any impediment that prevents me, or the Team from meeting the Sprint Goal?

The daily scrum is not a status report. If an impediment is surfaced during the daily scrum that is too big to resolve during the meeting, the team should coordinate outside of the meeting to address it. The Scrum Master is the team member responsible for removing impediments. In addition to the daily scrum, the team should spend 5-10% of its time looking ahead and refining the items at the top of the product backlog. This is called backlog refinement. It is not an official Scrum ceremony, but it is a best practice. During refinement, just as in sprint planning, the team focuses on the top of the product backlog to make sure those items are ready to be brought into the next sprint. At the end of each sprint, the team invites the stakeholders and customers to a demonstration of what it has completed. This ceremony, called sprint review, is designed to elicit actionable feedback from the stakeholders and customers, which the team can then incorporate into the product backlog. The demonstration produces a conversation between the team and the stakeholders about how to make the product better. The product owner incorporates the lessons learned during the conversation into the product backlog. This completes one of what is hopefully many inspect and adapt cycles. After sprint review, the team gathers for the final ceremony of the sprint, sprint retrospective. The retrospective is the team's opportunity to inspect and adapt its processes. A common way to structure a sprint retrospective is to have each team member answer the following questions:

1. What went well?
2. What could have been better?
3. What things can we try to improve in the coming sprint?

The discussion about what could have been better often leads to an analysis of what the underlying cause might be. A consensus about what improvement to make in the next sprint usually starts to emerge. This single process change is added to the next sprint's backlog and given a definition of done.

2.5.8. RUP

Rational Unified Process, maintained by IBM since 1998. The Rational Unified Process is a software engineering process. It provides a disciplined approach to assigning tasks and responsibilities within a development organization. Its goal is to ensure the production of high-quality software that meets the needs of its end users within a predictable schedule and budget. There are three central elements that define RUP:

1. An underlying set of principles for successful software development. These principles are the foundation on which the RUP has been developed.
2. A framework of reusable method content and process

building blocks. A family of method plug-ins defines a method framework from which you create your own method configurations and tailored processes.

3. The underlying method and process definition language. A unified method architecture meta-model that provides a language for describing method content and processes.

The Rational Unified Process captures many of the best practices in modern software development in a form that is suitable for a wide range of projects and organizations. Along with many others, it covers major practices:

1. Develop software iteratively.
2. Manage requirements.
3. Use component-based architectures.
4. Visually model software.
5. Continuously verify software quality.
6. Control changes to software.

2.6. Extreme Programming

Since 1999 is a lightweight, efficient, low-risk, flexible, predictable, scientific, and fun way to develop a software. eXtreme Programming (XP) was conceived and developed to address the specific needs of software development by small teams in the face of vague and changing requirements. Extreme Programming is one of the Agile software development methodologies. It provides values and principles to guide the team behaviour. The team is expected to self-organize. Extreme Programming provides specific core practices where

1. Each practice is simple and self-complete.
2. Combination of practices produces more complex and emergent behaviour.

A key assumption of Extreme Programming is that the cost of changing a program can be held mostly constant over time. This can be achieved with

1. Emphasis on continuous feedback from the customer
2. Short iterations
3. Design and redesign
4. Coding and testing frequently
5. Eliminating defects early, thus reducing costs
6. Keeping the customer involved throughout the development
7. Delivering working product to the customer

Extreme Programming involves

1. Writing unit tests before programming and keeping all of the tests running at all times. The unit tests are automated and eliminates defects early, thus reducing the costs.
2. Starting with a simple design just enough to code the features at hand and redesigning when required.
3. Programming in pairs (called pair programming), with two programmers at one screen, taking turns to use the keyboard. While one of them is at the keyboard, the other constantly reviews and provides inputs.
4. Integrating and testing the whole system several times a day.
5. Putting a minimal working system into the production quickly and upgrading it whenever required.
6. Keeping the customer involved all the time and obtaining constant feedback. Iterating facilitates the accommodating changes as the software evolves with the changing requirements.

i. AUP: Agile Unified Process is maintained since 2005 by Scott Ambler. “The agile unified process is a hybrid modelling approach created by Scott Ambler when he combined the Rational Unified Process (RUP) to agile methods (AM)” [6]. Scott Ambler works for the IBM Methods group as the practice leader for agile development (IBM, n.d.). By combining RUP to AM, Ambler created a solid process framework that can be applied to all sorts of software projects, large or small. Agile methods provided values, principles, and practices to AUP. The agile manifesto shows what these values and principles are. The manifesto describes four value statements for agile development. These values include individuals and their actions, delivering working software, customer collaboration, and responding to change (Sutherland & et al., 2001). The principles described in the manifesto include satisfying the customer through early and continuous software deliverables, welcoming change, developers and business collaborating throughout the project, building projects through motivated individuals, using the most effective means of conveying information like face to face.

When Ambler created the AUP, he centred the design around the following principles:

1. Most people won't read detailed documentation. However, they will need guidance and training now and then.
2. The project should be described simply in a few pages.
3. The AUP conforms to the values and principles described by the Agile Alliance.
4. The project must focus on delivering essential value rather than unnecessary features.
5. Developers must be free to use tools best suited to the task at hand, rather than to comply with an edict.
6. AUP is easily tailored via common HTML editing tools. Source: (Ambler, 2005).

ii. DAD: Disciplined agile delivery Supersedes AUP. Many organizations start their agile journey by adopting Scrum. However, Scrum is only part of what is required to deliver sophisticated solutions to your stakeholders. Invariably teams need to look to other methods to fill in the process gaps that Scrum purposely ignores. When looking at other methods there is considerable overlap and conflicting terminology that can be confusing to practitioners as well as outside stakeholders. To address these challenges, the Disciplined Agile Delivery (DAD) process decision framework provides a more cohesive approach to agile solution delivery. To be more exact, here is a definition “The Disciplined Agile Delivery (DAD) decision process framework is a people-first, learning-oriented hybrid agile approach to IT solution

delivery. It has a risk-value delivery lifecycle, is goal-driven, is enterprise aware, and is scalable." [3]

DAD is a hybrid approach which extends Scrum with proven strategies from Agile Modelling (AM), Extreme Programming (XP), Unified Process (UP), Kanban, Lean Software Development, Outside in Development (OID) and several other methods. Although DAD was originally developed by IBM, it is a non-proprietary, freely available framework that does not require IBM tooling in any way. DAD extends the construction-focused lifecycle of Scrum to address the full, end-to-end delivery lifecycle from project initiation all the way to delivering the solution to its end users. It also supports lean and continuous delivery versions of the lifecycle unlike other agile methods, DAD doesn't prescribe a single lifecycle because it recognizes that one strategy does not fit all.

iii. SAFe®: The Scaled Agile Framework is a freely revealed knowledge base of proven, integrated patterns for enterprise-scale Lean-Agile development. It is scalable and modular, allowing each organization to apply it in a way that provides better business outcomes and happier, more engaged employees [7]. SAFe synchronizes alignment, collaboration, and delivery for large numbers of Agile teams. It supports both software and systems development, from the modest scale of well under 100 practitioners to the largest software solutions and complex cyber-physical systems, systems that require thousands of people to create and maintain. SAFe was developed in the field, based on helping customers solve their most challenging scaling problems. It leverages three primary bodies of knowledge: Agile development, Lean product development, and systems thinking.

SAFe can be configured with the three or four organizational levels described below:

1. Team level – SAFe is based fundamentally on Agile teams. Each team is responsible for defining, building, and testing stories (small pieces of new functionality) from their backlog. Teams deliver value in a series of fixed-length iterations (also called sprints). Teams use a common iteration cadence to synchronize work with other teams; this allows the entire system to iterate simultaneously. Teams employ Scrum (primarily) or Kanban methods. Each of these methods is augmented by built-in quality practices. Many software quality practices are derived from eXtreme Programming, while hardware and system quality practices are derived from contemporary Lean product development practices.
2. Program level – SAFe teams are organized into a virtual program structure called the "Agile Release Train" (ART). Each ART is a long-lived, self-organizing team of Agile teams (typically 5 to 12), along with other stakeholders, that plan, commit, execute, inspect, and adapt together. ARTs are organized around the enterprise's significant value streams. They align teams to a common mission, provide architectural and user experience guidance, facilitate flow, and provide continuous objective evidence of progress.
3. Value Stream level – The optional Value Stream level supports the development of large and complex solutions. These solutions require multiple, synchronized ARTs, as well as stronger focus on solution intent and solution context. Suppliers and additional stakeholders contribute to this level as well. Pre-and Post-Program Increment (PI) planning inform the ARTs (and vice versa) of the Value Stream mission and objectives.
4. Portfolio level – The Portfolio level organizes and funds a set of value streams. The value streams realize a set of solutions, which help the enterprise achieve its strategic mission, as defined in part, by a set of strategic themes. The Portfolio level provides solution development funding via Lean-Agile budgeting, any necessary governance, and coordination of larger development initiatives that affect multiple value streams.
5. Foundation layer – The Foundation layer holds various additional elements that support development. Elements of the Foundation layer include: Lean-Agile Leaders, Communities of Practice, Core Values, Lean-Agile Mindset, and Principles.

iv. LeSS: Large-Scale Scrum provides two different large-scale Scrum frameworks. Most of the scaling elements of LeSS are focused on directing the attention of all the teams onto the whole product instead of "my part." Global and "end-to-end" focus are perhaps the dominant problems to solve in scaling. The two frameworks which are basically single-team Scrum scaled up are:

1. LeSS: Up to eight teams (of eight people each).
2. LeSS Huge: Up to a few thousand people on one product.

3. Conclusion

From the various dispensations highlighted above, the improvement in system development has been explosive. The trends observed in software engineering include finding and eliminating defects earlier in the development life cycle to cut costs and increase speed and efficiency. Elaborate, analyse, and verify the models before development. Coding, which is the heart of development is not given enough emphasis because without a methodology, then there won't be needing to codify. Testing is the gateway to check for defects before delivery. Limiting resources (mainly team) to accommodate budget. The emerging Methodologies depicts dynamism in how SDLC can be done better and faster. There is also tendency that many improvements will happen soon, because problems never finish, so also solutions.

References

[1] A. Dennis, B. Haley and R. Roth, Systems Analysis and Design, 5th ed. 2015.

[2] A. Scott, The agile unified process (AUP), 2005. [Online]. Available: http://www.ambysoft.com/unifiedprocess/agileUP.html. [Accessed: Nov 24, 2017].

[3] A. Scott, Going Beyond Scrum Disciplined Agile Delivery, 2013. [Online]. Available: https://disciplinedagileconsortium.org/Resources/Documents/BeyondScrum.pdf. [Accessed: Jan 2, 2018].

[4] B. Bruegge, Methodologies: Extreme Programming and Scrum "Introduction into Software Engineering" 2006. [Online]. Available: https://www1.in.tum.de/lehrstuhl_1/files/teaching/ss07/SE/SE2007_Lecture23.pdf. [Accessed: Dec 18, 2017].

[5] CASE Maker Inc. What is Rapid Application Development, 1997. [Online]. Available: http://www.iro.umontreal.ca/~dift6803/Transparents/Chapitre1/Documents/rad_wp.pdf. [Accessed: Dec 18, 2017].

[6] C. Ioannis, P. Stravos and P. Eleni, Using the agile unified process in banking, Apr, 2010. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5232801. [Accessed: Dec 18, 2017].

[7] D. Leffingwell, Overview of the scaled Agile Framework for Lean software and Systems Engineering, 2016. [Online]. Available: https://www.iconagility.com/docs/SAFe-Fact-Sheet.pdf. [Accessed: Feb 3, 2018].

[8] E. Geoffrey, Global Business Information Technology, 2004.

[9] Freetutes. Object Oriented Methodology, 2017. [Online]. Available: http://www.freetutes.com/systemanalysis/sa2-object-oriented-methodology.html. [Accessed: Feb 5, 2018].

[10] IBM Software Group Rational Unified Process: A Best Practices Approach, 2012. [Online]. Available: http://www.eecg.toronto.edu/~jacobsen/courses/ece1770/slides/rup.pdf. [Accessed: Feb 4, 2018].

[11] LeSS Company B. V. LeSS Framework, 2016. [Online]. Available: https://less.works/less/framework/index.html. [Accessed: Nov 24, 2017].

[12] M. Elliott and W. Scacchi, "Free software development: cooperation and conflict in a virtual organizational culture", in Koch, S. (Ed.), Free/Open Source Software Development, Idea Group Publishing, Hershey, PA, pp. 152-72, 2004.

[13] M. James Grier, Living systems: basic concepts. Behavioural Science 10, 1965.

[14] M. L. Tushman and L. Rosenkopf, "Organizational Determinants of Technological Change: Toward a Sociology of Technological Evolution," Research in Organizational Behaviour, Vol 14: pp. 311-347, 1992.

[15] S. James, Analysis and Design of Information Systems. New York: McGraw-Hill, 1989.

[16] S. Herbert, "The Architecture of Complexity," Proceedings of the American Philosophical Society 106: pp. 467-482, reprinted in idem. (1981) The Sciences of the Artificial, 2nd ed. MIT Press, Cambridge, MA, pp. 193-229, 1962.

[17] ScrumInc The Basics of Scrum An introduction to the framework, 2014. [Online]. Available: https://34slpa7u66f159hfp1fhl9aur1-wpengine.netdna-ssl.com/wp-content/uploads/2014/06/The-Basics-of-Scrum.pdf. [Accessed: Nov 24, 2017].

[18] S. Jeff, Manifesto for agile software development, Feb. 2, 2001. [Online]. Available: http://agilemanifesto.org/. [Accessed: Dec 12, 2017].

[19] W. Jensen, Structured Programming, 1981. [Online]. Available: https://pdfs.semanticscholar.org/3a86/903b274c6b24217885b638a74521032a708e.pdf. [Accessed: Dec 27, 2017].

[20] Wiki books Introduction to Software Engineering/Process/Methodology, 2018. [Online]. Available: https://en.wikibooks.org/wiki/Introduction_to_Software_Engineering/Process/Methodology. [Accessed: Feb 19, 2018].