FARUL3S: a New Framework for Agile Regulated Ultra-Large-Scale Software Systems

This paper was downloaded from TechRxiv (<https://www.techrxiv.org>).

LICENSE

CC BY 4.0

SUBMISSION DATE / POSTED DATE

08-07-2021 / 25-01-2022

CITATION

KSIKSI, Amira; Lazzez, Onsa; Aidi, Maher; Ltifi, Hela; Wali, Ali; Hamdani, Tarek M.; et al. (2021): FARUL3S: a New Framework for Agile Regulated Ultra-Large-Scale Software Systems. TechRxiv. Preprint.
<https://doi.org/10.36227/techrxiv.14935392.v4>

DOI

[10.36227/techrxiv.14935392.v4](https://doi.org/10.36227/techrxiv.14935392.v4)

FARUL3S: a New Framework for Agile Regulated Ultra-Large-Scale Software Systems

**Amira Ksiksi ^{a,*}, Onsa Lazzez ^a, Maher Aidi ^b, Hela Ltifi ^{a, c}, Ali Wali ^a, Tarek M.
Hamdani ^a, Abdulrahman M. Qahtani ^d, Omar Almutiry ^e, Habib Dhahri ^e, Adel M.
Alimi ^{a, f}**

^a REsearch Groups in Intelligent Machines (REGIM Lab), University of Sfax, National Engineering School of Sfax (ENIS), BP 1173, Sfax, 3038, Tunisia.

^b Quality department, Higher Institute of Industrial Management of Sfax (ISGIS), University of Sfax, BP 1164, Sfax, 3021, Tunisia.

^c Computer Science and Mathematics Department, Faculty of Sciences and Techniques of Sidi Bouzid, University of Kairouan, Tunisia.

^d Department of Computer Science, College of Computers and Information Technology, Taif University, P.O.Box. 11099, Taif 21944, Saudi Arabia.

^e College of Applied Computer Science, King Saud University, Riyadh, Saudi Arabia.

^f Department of Electrical and Electronic Engineering Science, Faculty of Engineering and the Built Environment, University of Johannesburg, South Africa.

* **Corresponding author.** Email address: amira.ksiksi@regim.usf.tn

Email addresses: onsa.lazzez@regim.usf.tn (O. Lazzez), maher.aidi@isgis.rnu.tn (M. Aidi), hela.ltifi@ieee.org (H. Ltifi), ali.wali@ieee.org (A. Wali), tarek.hamdani@ieee.org (T. M. Hamdani), amqahtani@tu.edu.sa (A. M. Qahtani), oalmutiry@ksu.edu.sa (O. Almutiry), hdhahri@ksu.edu.sa (H. Dhahri), adel.alimi@regim.usf.tn (A. M. Alimi)

Abstract

The Ultra-Large-Scale Software (ULSS) systems development challenges today's top-down software management and development approaches. Northrop et al. (2006) revealed three broad areas of challenges [1]. To deal with those challenges, they propose an interdisciplinary portfolio of research. In particular, we address the design and evolution challenge by focusing on the design area of research.

In order to regulate the ULSS systems, we propose the Framework for Agile Regulated Ultra Large Scale Software System (FARUL3S). The FARUL3S is a user-centered bottom-up solution that aims at combining the complex adaptive system, the financial economics as well as the engineering systems design. In this paper, we provide a detailed description of the FARUL3S steps.

Our Framework aims to regulate and constrain the ULSS systems by generating a system Design Rule Hierarchy so it can be used to constrain the entire system design. As a result, we prove that the more new projects are designed, the more the added value is improved.

Key words: User-centered, regulation, bottom-up, Ultra-Large-Scale Software System, agile methodology, modularity, Design Structure Matrix.

1. Introduction

Due to the exponential growth of the software systems, the adoption of traditional software engineering approaches face challenges in the Ultra Large-Scale Software (ULSS) systems development. Northrop et al. revealed three broad areas of challenges and proposed an interdisciplinary portfolio of research to deal with those challenges [1]. In particular, we address the design and evolution challenge by focusing on the design area of research.

According to the Domain Driven Design (DDD), large models must be divided into different Bounded Contexts and be explicit about their interrelationships. Thereby, a domain must be split into sub-domains small enough to be grasped [2]. Whereas, the Smart City system covers diverse interoperable domains, which make the DDD an inadequate solution for the entire Smart City system development. By considering the Smart City system as an ULSS system, we need new perspectives to deal with its complexity among all the dimensions. The real challenge is to regulate the ULSS system as a city rather than be engineered as a building [1]. That's how the scale changes everything [1],

In this paper, we present a new Framework for Agile Regulated Ultra Large-Scale Software System (FARUL3S) which is a user-centered solution that aims at combining the complex adaptive system, the financial economics as well as the engineering systems design [1]. The aim of this paper is to illustrate the FARUL3S steps and its adoption in the software system management and development processes. In fact, there are some design steps that are already automated. For example, Simon Tool [23, 35] supports the design modeling and analysis framework, Minos Tool [25] as a refactored version of Simon, supports ACN modeling and a set of modularity analysis, such as the automatic derivation of the DSM, and Titan Tool which is a DSM viewer [26, 36]. To ensure the smooth running of the FARUL3S steps, we focus on illustrating the different framework steps so we provide a description for each phase as it will be performed manually.

The next section provides the related works in which we introduce the fundamental concepts behind our Framework. Subsequently, in section 3, we propose our Framework FARUL3S and present an illustration of its different steps. Then, section 4 we provide a Net Option Value (NOV) based analysis of our Framework adoption. Finally, we conclude with a discussion of our contribution followed by future research directions.

2. Background

IBM defined the innovative systems of smart cities as the use of information and communication technology to sense, analyze and integrate the key information of core systems in running cities. The smart cities can provide innovative intelligent services and urban management thereby promoting a more efficient, harmonious and convenient process for modern cities [3].

The exploration of user needs by gathering user feedback is important for a successful innovation development. Therefore, the Open Living Lab, where everyone can participate [4], must be adopted in the management and the development process of the smart cities systems.

Despite the studies provided by the previous literature, which show that the living lab can be a conceptualization, a context, or a methodology, a multiplicity of approaches, an involvement of users and multiple stakeholders, as well as the importance of the real-life environment, are commonly addressed by the living lab definitions [5–7].

To facilitate the identification of the actors that can drive the innovation, Leminen, Westerlund and Nyström proposed four types of living lab which are: utilizer-driven, enabler-driven, provider-driven and user-driven [8]. Furthermore, based on the customized tools and iterative innovation processes, a new typology was introduced: linearizer, iterator, mass customizer and tailor [9]. Bergvall-Kåreborn et al. [10] propose the FormIT method that facilitates co-creation in the living lab. Furthermore, Tang and Hämmäläinen [11] define a process model consisting of five iterative phases: requirements, co-design, prototyping, test and tracking, and commercialization.

Generally, a living lab aims to create and develop a creative innovation with high-quality that provides the needs of a specific customer segment [12] by their

involvement in the real-life settings [4]. Nevertheless, the living lab lacks a rapid reactivity to the end-users requirements creep. On the other hand, a structural focus on users is needed in the agile methodologies [13].

To provide agility during the project development, an agile methodology must be adopted. SCRUM [14], the most adopted agile methodology, lacks a structural focus on usability. As a result, The U-SCRUM [15] appears to deal with the SCRUM barriers by making more focus on user feedback. On the other hand, Lean UX [16, 17] attempts to build a better experience by adding a user experience designer as a central role in the development process. The Lean UX consists of four phases: (1) declare assumptions, (2) create a minimum viable prototype (MVP), (3) run an experiment, and (4) get user feedback which can be translated into new insights and assumptions. Frequently, by using U-SCRUM [15] and Lean UX [16], the required work is failed to be performed, as the UX designers tend to work on many projects at the same time. Also, the product testing is usually performed only with internal people rather than representative user profiles.

Fig. A.1: Framework for Agile Living Lab projects (FALL) [18]

Hence, merging the agile development methodologies with living labs can provide the needed agility in the living lab. Coenen Robijt [18,19] proposed the Framework for Agile Living Lab (FALL), that combines the living lab approach with agile methodologies. Generally, the FALL, as represented in figure A.1, consists of three main phases: 1) the Problem formulation, 2) the BIEL and 3) the formalization of the learning phase. The problem formulation phase contains the set of user roles followed by the map and prioritization of assumptions. As in Lean UX, the high priority hypothesis, that has low maturity and high risk, must be the focus of the first MVP [16]. The BIEL phase, which can be considered as SCRUM sprints, contains the build, the intervention and the evaluation of successive MVP prototypes. Finally, in the formalization of the learning step, all that was learned is reflected in a suitable format for consumption [18, 19].

Fig. A.2: A detailed representation of FALL

To explain the Framework more, the figure A.2 represents an illustration of each phase. The detailed FALL steps are defined as follows: (1) Set of user roles, which attributes the responsibilities to open living lab members, (2) map and validating assumptions by filling the Innovatrix framework through probing questions in an Innovatrix workshop, (3) map user-stories via organizing a story mapping workshop to ask the end-user to write stories, (4) transfer user-stories into backlog, (5) draft the system architecture, (6) quantify the needed effort for each sprint, (7) define and perform iterative sprints, (8) test results with end-users which reflect new insights, and (9) contribute to literature in the field of learning from the prototype as final product in the end of the project step.

The FALL presents a solution for innovative software systems engineering that meets the Domain Driven Design (DDD) requirements. In the DDD, every bounded context, which is the boundary within a model, has its own ubiquitous language and domain

model [2]. Moreover, there is the context map that reflects the current situation of the relationships among project's bounded contexts [2].

As represented in the figure A.3, the DDD layered architecture contains four levels of abstraction which are the user interface layer, the application layer, the domain layer and the infrastructure layer [2]. Each layer depends only on layers of the same level as well on the ones beneath it [2].

Layered Architecture

Fig. A.3: DDD Layered Architecture [2]

The DDD is a way of looking at software from top-down. Unfortunately, this layered hierarchy cannot deal with ULSS system complexity due to the system domain's diversity and its ultra large size.

We need to look at the ULSS system from bottom-up to regulate the system model and constrain the system using architectural agreements and other rules, which create a decentralized and parallel exploration of design spaces.

The design spaces are represented with Design Structure Matrix (DSM), which is a popular tool for abstract system modeling [2, 20]. The DSM is a square matrix in which rows and columns represent the design variables.

To represent the dependencies among design decisions, a mark in a cell means that the design decision corresponding to its row depends on the one that corresponds to its column. As the column elements have the same order with the row elements, the diagonal cells cannot be marked.

Baldwin and Clark introduce the modular structure for computer systems by using the DSM and the design rule theory. The highest left cells in the DSM represent the design rules as they can affect any other design parameters, whereas no one can affect it [20]. By using the DSM, the design rules and the real option theory, Baldwin and Clark account for the influence of the modularity in design value and how it can be modeled as real options to analyze design decisions and maximize the added value [20].

To structure design space and decision-making, we need to structure design parameters as well as their dependencies. So we need to cluster design parameters into proto-modules, followed by the application of the splitting operator to decouple otherwise coupled modules [20].

The substitution operator aims to decide whether to adapt or improve a hidden module by maximizing the added value based on decision replacements [20].

Specifically, it deals with each independent module separately, and aims at calculating the Net Option Value to determine the maximum expected number of experiments to maximize the gain [20]. In that sense, modularity in design provides a portfolio of options that is worth more than an option on a portfolio [21].

The ***splitting*** and ***substitution*** are two of the six modular operators introduced by Baldwin and Clark. Indeed, the other operators aim to modify already existing designs: ***Augmentation*** to add a module to the system, ***exclusion*** to remove a module, ***inversion*** to standardize the common design elements and create new design rules, and ***porting*** to transport a module to another system.

To deal with the software system design, Parnas contribute the information hiding approach to decouple design decisions into modular structure so it can be changed independently [22]. Sullivan et al. extend Baldwin and Clark design rule theory to Parnas's notion of information hiding which proved that the economic prediction of the model fits the Parnas conclusions and prove that information hiding modularity (IHM) creates valuable options to switch from a hidden module to another one [21] and prove also that information-hiding design is twice as valuable as strawman design [21].

Furthermore, they reveal the Baldwin and Clark theory gap which is the lack of a suitable representation of the environment in which the software is deployed, as the environment can affect its design value [21]. Sullivan et al. extend the DSM modeling

framework to cover the environment parameters by introducing the Environment and Design Structure Matrices (EDSM).

The Design rule decouples design decisions to decompose systems into component parts. Further, it aims to identify the design space structure to be explored by competitive and cooperative organizations to design its individual parts [2, 20]. Once the modules are decoupled, by breaking the cyclical dependencies, the substitutions of modules can be done in parallel and in a decentralized way as it does not affect the other modules [20].

To design software systems, Cai introduced the formal design modeling and analysis framework [23]. This framework involves three core models: the augmented constraint network (ACN), the Design Automaton (DA) and the Pair-wise Dependence Relation (PWDR) [23, 24]. It drives the DA from the ACN and drives the PWDR from the DA as represented in figure A.4 [23, 24]. An illustration of the modeling framework design steps will be presented in the next section.

Fig. A.4: design modeling and analysis framework [23]

To deal with the continued growth of software projects scale, Wong et al. present the Design Rule Hierarchy (DRH) to identify the modules that can be parallelized [25].

The DRH is derived from the DSM as it provides a clear representation of the coordination between component parts.

In software system design, dependent modules represent a layered architecture [25].

The modules that belong to the same layer must be independent, hence parallelizable.

However, dependencies among layers or within a module need to be coordinated so it can be developed concurrently [25].

To improve the software architecture recovery, Cai et al. contribute an architecture recovery approach to recover the design rules and then the modules related to these design rules. And based on this approach, they developed an Architectural Design rule hierarchy (ArchDRH) algorithm to deal with the system's complexity [26]. The ArchDRH can be determined by applying three main steps: 1) apply the DRH, 2) identify the conceptual modules to regroup the modules that follow the same concept, 3) apply the ArchDRH-Family algorithms to decompose large modules [26]. For example, the ArchDRH-Re algorithm aims at separating the control modules. Control module is the one that depends on the majority of the components and does not have dependents [26].

To quantify the benefits of modularity in connection with the level of design flexibility, Baldwin and Clark invest the real option theory [20]. The microeconomics field provides a set of software engineering economics analysis techniques, such as software cost estimation models [27, 31, 32]. Basically, the cost estimation is critical for investment decision-making. To invest under uncertainty, Boehm [27] pioneered the Bayesian decision theory techniques to perform prototyping decision making. This approach lacks the flexibility which constrains an immediate design-choice that means there is a one period time to invest under uncertainty.

Fortunately, the real options theory captures that the flexibility of waiting for better information can add a significant value [28]. Therefore, Chalasani et al. propose a

delayable investment under uncertainty approach which postpone design decisions as well as prototyping based on real options theory [29].

The future uncertainty is modeled with a multi-period discrete event tree of finite depth N , with N the maximum future time steps number [29]. The present time is represented with the root node which is the node at depth 0.

The nodes that belong to a specific depth represent the different system states at this time step [29].

Sullivan et al. Extend the theory to cover the hiding information [22] criterion of software design [28]. They present a theory to evaluate and analyze the information hiding in software design. Further it can make decisions when to delay investment under uncertainty [28]. The real options lead to better understanding of new insights in software design by accounting the flexibility value in terms of real options [30].

Sullivan et al. reveal the gap in Baldwin and Clark theory which is the lack of representing the environment variables that have an effect on the design value [21]. Therefore, they extend the theory to represent the environment parameters. And they prove that the environment parameters as a variant on design affect the added value [21].

By linking the information hiding and spiral processes which are two software design principles, Sullivan et al. developed an option-based rethinking approach for software design and engineering. The linkage is done in terms of the decision flexibility

accounting using the real options value [30]. Therefore, information hiding and incremental design and implementation using an agile methodology are two effective principles of software design that serve to maximize the value of decision criteria to invest under uncertainty as the agile methodology decreases uncertainty risks [30].

3. Proposed Solution

The smart city, as an intensive investigation of the ICT to facilitate the citizen's daily-life tasks, is characterized by its complexity, high distribution, heterogeneity as well as its unpredictable work environment. The smart cities must be modeled as an Ultra-large scale (ULS) systems due to its expected complexity of computation, information and the required communication systems, and the ultra large size according to multiple measures, such as “the number of lines of code; number of people employing the system for different purposes; amount of data stored, accessed, manipulated, and refined; number of connections and interdependencies among software components; number of hardware elements; etc” [1].

The reductionism (parts thinking) philosophy, which is adopted by the traditional software engineering approaches, is unable to deal with the exponential growth of the smart cities systems [33]. As the smart cities systems are greater than the sum of its elements, the system thinking (holism) paradigm must be adopted to face the challenging characteristics. The holism philosophy considers the common objectives of the entire system are outside the sum of the individual's goals, because of the behaviors and characteristics that appear from the interaction of its parts [33].

To ensure the adaptation and the interaction between the different individual parts, the whole smart city system must be regulated rather than engineered [1]. Therefore, we need a new innovative approach to change the perspective from the traditional, top-down engineering to the bottom-up regulation of the decentralized and complicated systems.

To provide the prerequisite regulation, we propose the Framework for Agile Regulated Ultra-Large-Scale Software System (FARUL3S). The FARUL3S is a user-centered and bottom-up solution that aims at combining the complex adaptive system, the financial economics as well as the engineering systems design. As in the smart cities, the functionalities are not clear from the beginning, the development process is chosen to be incremental to be able to add new features at each round.

Moreover, our framework takes into account the open living lab to integrate the end-user during the development process. Furthermore, to maximize the added value and to constrain the system layered hierarchy, the FARUL3S design process constraints design space exploration by using the design rule theory, which creates modularity structure for parallel and decentralized substitution as well as development.

By clustering the system into loosely coupled modular structures, the design rules give the competitive and cooperative organizations the incentives to work on the software system modules as component parts in parallel and concurrently.

The figure A.5 represents the FARUL3S architecture which combines the FALL, the software engineering economics and the design modeling.

Fig. A.5: Framework for Agile Regulated Ultra-Large-Scale Software System (FARUL3S)

In this section we provide an illustration of the different FARUL3S steps. Starting with the set of user roles step, There are 8 user roles that must be adjusted in the FARUL3S framework as represented in the FALL framework: 1) a process manager

that guides the team, 2) a user researcher which helps the end-user to collaborate in the FALL and keep the story backlog up to date, 3) a researcher to obtain input from users, 4) an architect to create the systems architecture, 5) a UX designer who creates the GUI design of the system, 6) A developer to develop the story backlog, 7) an end-user which brings domain oriented knowledge to the team via usability testing and co-design, and 8) a stakeholder that holds higher-level interest than end-users by operating from a commercial, social or academic point of View [18,19].

The next step is the mapping and validation of projects. In this step, the open living lab organized a workshop to integrate the end-user and the different stakeholders to propose their needed daily-life services. The gathered insights are then grouped into different platforms according to the domain that it belongs to.

By applying the DDD strategic tool, each platform represents a separate domain and each project represents a bounded context that requires its own ubiquitous language and architecture to deal with the sub-domain problem space.

Fig. A.6: Smart city system platforms and services

By taking the smart city system as a use case, the figure A.6 identifies the different project's taxonomy among platform domains based on the Bounded Contexts and Context Maps. By the way, those results are obtained from a workshop organized by the REGIM Lab. In fact, the software engineering methods did not fit the domain's diversity and the interoperability between the different domains which made this hierarchy face a big challenge in its development.

The third step is the project selection, in which each organization performs its own requirements such as resources and knowledge. In this step, competitive and cooperative organizations and even open living labs can select a project to work on. Once the project is selected, the organization starts the mapping and validation of assumptions. To perform this step, an Innovatrix workshop is organized to fill the Innovatrix framework. After the gathering of inputs through probing questions by the facilitator, the innovator fills the input and then allocates for each input a status: New insights (no input), Assumption (the input not yet validated), Invalidated (the input is not supported), Validated (the input has been already validated) [34]. The most important assumptions considered as “key uncertainties” [34].

In the user stories mapping phase, a workshop is organized and led by the user researcher. In this workshop, all participants are asked to write user stories which are then gathered, grouped and prioritized by filling the must have, should have, could have and won't have columns of the MOSCOW method [18]. The high-impact

user-stories are selected to be the focus of the first backlog in the transfer of user-stories into a backlog step.

As soon as the backlog is ready, we proceed to the design stage. Basically, in our solution, we start with the design modeling and analysis framework [23] followed by DRH and ArchDRH approaches. Then, we perform the time and cost estimation based on numerical DSM for investment decision-making.

For more details, we provide an illustration of the different design steps. The first step is the class diagram performance, followed by the extraction of the ACN. Basically, the UML class diagram takes into account the system dominance relations that are obtained from the system DRH step which is the final step of the design stage. For the first project development, there are no system dominance relations. Also, to ensure the smooth running of the framework, each organization must split the identified design rules of a project into smaller ones to cover as many higher levels of the hierarchy as possible. Moreover, the design rule level identification is based on the bounded contexts so we can identify the different domain models. On the other hand, the ACN consists of the Constraint Network (CN), the dominance relations and the cluster set. The CN is used to model design dimensions and external conditions. On its part, the dominance relation data structure is used to deal with the modularity analysis issues [23].

The CN consists of variables and constraints. On one hand, we have the variables, which are divided into two parts: the variables which represent the design parameters that will be used for design decision-making, and the values which represent the decisions made [23]. On the other hand, there are the logical constraints that are used to represent the required relations [23].

As an example, we model just some variables and constraints of the Smart Social project that we have already developed.

```
1: PromotionService : {orig, other};
2: PublicityService : {orig, other};
3: User : {orig, Admin, Supervisor, Professional, Trader, Citizen, other};
4: SignupService : {orig, other};
5: SignupServiceImpl : {orig, other};
6: SignupController : {orig, other};
7: LikeService : {orig, other};
8: LikeServiceImpl : {orig, other};
9: SignupServiceImpl = orig => SignupService = orig && UserRepository = orig &&
TokenRepository = orig && User = orig && City = orig && Country = orig;
10: CitizenPromotionServiceImpl = orig => User = Citizen &&
CitizenPromotionService= orig && PromotionService = orig;
```

```
11: LikeServiceImpl = orig => LikeService = orig && UserService= orig &&  
PromotionService = orig;
```

The instructions from 1 to 8 models the variables and its different possible values choices. In addition, the three final instructions represent the logical constraints. For example, line 10 indicates that the choice of 'orig' for CitizenPromotionServiceImpl is valid if and only if the CitizenPromotionService and the PromotionService are equal to 'orig' and the User is 'Citizen'.

The second step of the ACN is the dominance relations definition [23]. Indeed, the dominance relations reveal the variables that dominate the system design hierarchy. Take the following instructions as examples:

1: (TraderPromotionServiceImpl, NewsService)

2: (LoginController, LoginService)

3: (PublicityServiceImpl, PublicityService)

4: (LikeServiceImpl, LikeService)

The first line means that the design variable LikeServiceImpl depends on the LikeService but it cannot affect it. The same thing for the other relations.

Finally, to complete the ACN, we need to define the cluster set which can be used to select the optimal modularity structure. The same design can be clustered using different methods. Each method represents a different stakeholder's view [23].

Once the ACN of the software system is completed, we proceed to the derivation of the DA. The DA is a state-machine-based model which supports a set of economic and evolvability analysis. To reach the DA, we need to perform the splitting approach. On its part, the splitting approach starts with the Conjunction Normal Form (CNF) extraction from the logical constraints [23]. For more illustration, we take the same logical constraints instructions used above as examples to illustrate the CNF extraction.

1: $(\neg \text{SignupServiceImpl} = \text{orig} \vee \text{SignupService} = \text{orig}) \wedge$

2: $(\neg \text{SignupServiceImpl} = \text{orig} \vee \text{UserRepository} = \text{orig}) \wedge$

3: $(\neg \text{SignupServiceImpl} = \text{orig} \vee \text{TokenRepository} = \text{orig}) \wedge$

4: $(\neg \text{SignupServiceImpl} = \text{orig} \vee \text{User} = \text{orig}) \wedge$

5: $(\neg \text{SignupServiceImpl} = \text{orig} \vee \text{City} = \text{orig}) \wedge$

6: $(\neg \text{SignupServiceImpl} = \text{orig} \vee \text{Country} = \text{orig}) \wedge$

7: $(\neg \text{CitizenPromotionServiceImpl} = \text{orig} \vee \text{User} = \text{Citizen}) \wedge$

8: $(\neg \text{CitizenPromotionServiceImpl} = \text{orig} \vee \text{CitizenPromotionService} = \text{orig}) \wedge$

9: $(\neg \text{CitizenPromotionServiceImpl} = \text{orig} \vee \text{PromotionService} = \text{orig}) \wedge$

10: $(\neg \text{LikeServiceImpl} = \text{orig} \vee \text{LikeService} = \text{orig}) \wedge$

11: $(\neg \text{LikeServiceImpl} = \text{orig} \vee \text{UserService} = \text{orig}) \wedge$

12: $(\neg \text{LikeServiceImpl} = \text{orig} \vee \text{PromotionService} = \text{orig}) \wedge$

Without loss of generality, Cai suppose that each variable in the ACN is implicated in at least one clause [23]. We can define clauses based on ubiquitous languages involved in the bounded context. To define the variables clause, we need to focus on the first variable of the instruction. For example, in line 10, we take the 'LikeServiceImpl' and look in all the other instructions for those that start with a variable that belongs to the same bounded context. And then, we define the clause as all variables that exist in the gathered instructions. Furthermore, the clauses must be small enough to be grasped. After that, each clause is modeled separately into a directed complete subgraph which is also called a CNF graph [23].

In this graph, the variables are represented with nodes. By default, each node is connected to every other node. According to the dominance relations, for each variable pair (V_i, V_j) belongs to the dominance relations, the edge $\langle V_i, V_j \rangle$ must be removed from the graph [23].

Fig. A.7: partial CNF graph

As shown in the figure A.7, the dotted lines correspond to the edges to be excluded so we obtain the directed graph.

According to Cai [23], to construct the condensation graph G^* which is a Directed Acyclic Graph (DAG), we need to aggregate the strongly-connected components into one node as represented in figure A.8.

Fig. A.8: Condensation graph

Furthermore, Cai [23] defines a technique to determine the number of sub-ACNs. By focusing on the lower level of the condensation graph, we can identify the minimal elements which are V3, V4 and V5 in this example. Hence, the number of the sub-ACNs is equal to the number of minimal elements which is three in this example. To construct the variable set, we cluster all the nodes that end with one of the minimal elements of G^* [23].

Next, we construct the set of constraints for each sub-ACN. The constraint set contains the CNF clause if the variable set of the sub-ACN contains all the CNF clause variables [23]. By taking the same example of constraints, we got the results below:

- 1: (!SignupServiceImpl = orig || SignupService = orig)&&
- 2: (!SignupServiceImpl = orig || UserRepository = orig)&&
- 3: (!SignupServiceImpl = orig || TokenRepository = orig)&&
- 4: (!SignupServiceImpl = orig || User = orig)&&
- 5: (!SignupServiceImpl = orig || City = orig)&&
- 6: (!SignupServiceImpl = orig || Country = orig)&&
- 7: (!CitizenPromotionServiceImpl = orig || User = Citizen)&&
- 8: (!CitizenPromotionServiceImpl = orig || CitizenPromotionService = orig)&&
- 9: (!CitizenPromotionServiceImpl = orig || PromotionService = orig)&&
- 10: (!LikeServiceImpl = orig || LikeService = orig)&&
- 11: (!LikeServiceImpl = orig || UserService = orig)&&
- 12: (!LikeServiceImpl = orig || PromotionService = orig)&&

The instruction in line 1 for example, means that SignupServiceImpl is equal to 'orig' if and only if the SignupService is equal to 'orig'. Then, we construct the sub-ACN dominance relation, by selecting the ones that contain its variables from the whole ACN. Subsequently, for each sub-ACN, we create a separate partial DA.

The DA states represent the different possible design states. To satisfy the constraints, a change in a design decision can result in a transition from one state to another one. The representation of the entire system DA is impractical due to its exponential growth

in relation to the number of variables [23]. That's why we split the whole ACN into smaller sub-ACNs that will be solved separately, and we integrate the only needed results to reach the desired answer in a more efficient way.

To derive PWDR from the DA, we need to remove all the incompatible states from each sub-DA [23]. A PWDR is a useful model behind the creation of influential design. A pair (x, y) belongs to the PWDR if and only if y is affected by x [23].

Take as example the following instructions:

- 1: (UserService, SignupServiceImpl)
- 2: (UserService, LoginServiceImpl)
- 3: (PromotionRepository, PromotionCitizenServiceImpl)

To derive the EDSM, we use the PWDR's obtained results to populate the matrix. So the EDSM cells are marked according to the PWDR. In addition, we select a clustering from the cluster set to order the matrix columns and rows [23]. As a result we obtain a modular hierarchy.

The next step is to cluster the software dependency structure into a DRH that fits Parnas's and Baldwin and Clark's modularity definition [25]. A DRH is a directed acyclic graph (DAG) where each vertex models a task; each task is defined as a set of design decisions that should be made together to identify the independent modules to be implemented concurrently [25]. The top of the hierarchy contains the most financial design rules that dominate the rest of the system decisions [25].

The DRH takes as input the condensation graph and the set of sub-ACNs. The approach starts with splitting each region of sub-ACNs intersection into a separate combination as illustrated in figure A.8.

Subsequently, we build a new graph H that contains the identified regions. Finally, we populate the H edges according to the ones in the condensation graph C [25]. As a result, we obtain the DRH as shown in figure A.9.

Fig. A.9: Partial hierarchy Graph

To derive the ArchDRH from the DRH, we start with the aggregation of the modules that pursue the same concept into a single conceptual module [26]. As a final step, the ArchDRH-Family aims to apply clustering techniques such as ArchDRH-Re which is used to separate control elements. A control module is known as a module that does not have dependents but depends on a majority of other elements within a layer [26].

Once the DSM is clustered using the ArchDRH-Re, it can be decomposed using ArchDRH-[3rd] and ArchDRH-Split algorithms [26].

On the other side, the derivation of the numerical DSM can be performed concurrently with the derivation of the DRH as these two steps are independent.

Due to the coupled design decisions, the software development lacks the uncertainty of the task duration. Smith et al. [37] propose a markov chain methodology that takes as input the numerical DSM and transforms it into markov chain stage tasks. The numerical DSM contains in its diagonal cells the required period of time to perform each task. On the other hand, the off-diagonal cells contain the likelihood of the rework probability that represent the strength of task coupling [37]. To automate the process, Titan et al. developed a fuzzy duration model for serial coupled iterative design tasks and as a result the model returns the fuzzy duration of each stage task separately [38]. Based on the expected duration, we determine the expected cost using the adequate cost estimation model [27, 31, 32] that meets the software system flexibility.

To decide whether to invest or not, we need a delayable investment under uncertainty approach that can postpone design decisions as well as prototyping to maximize the gain [29]. If the decision is to invest directly, the DRH of the current design must be added to the system DRH. By default, the system DRH does not contain any constraint. For each new project under development, the system DRH must be updated based on its DRH by adding the new constraints in the suitable level of the

entire system hierarchy as shown in figure A.10. Whereas, if the decision is to delay the investment, we need to wait for better information to improve the added value [29].

The DRH of the system aims at providing more guidance for the teams that are working on the different building blocks of the system by constraining the design space exploration.

Fig. A.10: Regulated system hierarchy

Once the investment decision is made and the system DRH is updated, we organize a sprint planning meeting to identify the backlog. Thereafter, the team members perform the sprint backlog by transferring it into a functional prototype among the iterative sprints. As in SCRUM, the development process of a sprint takes 2 to 4 weeks followed

by the sprint review to inspect the prototype and identify what is not well-functioning to emphasize more on it at the next sprints; and the sprint retrospective that assists to find ways to improve.

Next to the sprint backlog development, we need to perform a test of the prototype with the end-user. The test phase assesses the delivered prototype by gathering user feedback which reflect new insights that can make changes in the next backlog. To keep the backlog updated, the iterative process starts from the transfer of the user-stories to the backlog. When the objective is met, the final phase which is the end of the project is reached. In this phase, the learning from the development of the prototype must contribute to the domain literature.

Finally, if an organization wants to start a new service development or even to improve the developed one, it must start the process from the project selection phase.

4. NOV-based analysis

In this section, we provide a NOV-based analysis of the system design regulation compared with the traditional software engineering results. By the way, only the substitution options are addressed in this paper.

The NOV represents the expected payoff once the option is substituted optimally based on the created candidate replacements. According to Baldwin and Clark, the sum of the NOV of the n splitted modules within a project increase its base value S_0 [20, 21] as following :

$$V = S_0 + NOV_1 + NOV_2 + \dots + NOV_n \quad (1)$$

Moreover, Baldwin and Clark provide a model for calculating NOV [20, 21].

$$NOV_i = \max_{k_i} \{ \sigma_i n_i^{1/2} Q(k_i) - C_i(n_i)k_i - Z_i \} \quad (2)$$

For a given module i , $\sigma_i n_i^{1/2} Q(k_i)$ represent the expected benefit to be gained when k_i independent experiments generate the best positive-valued candidate [21]. $C_i(n_i)k_i$ determine the cost to run k_i experiments. $Z_i = \sum_{j \text{ sees } i} c_i n_j$ is the cost to replace the module i based on the number of the modules that directly depend on it. The max selects the number of experiments k_i that provides the maximum gain value reached for the module i . We assume that the $Q(k)$, which represents the expected value of k independent best draws from the standard normal distribution, are positive.

The choice of the maximum technical potential $\sigma=2.5$ is based on [21]. In our analysis results, we ignore the base value and focus only on modularity.

As use cases, we chose to work on the two projects Smart Social and Smart Education that belong to the same platform of the smart city system. Starting with the smart social project, as the system design rules hierarchy is initially empty, we split the obtained design rules into smaller ones to cover the platform and system level of the design rules hierarchy.

We choose to work with the same environment parameters that were defined by Sullivan et al. [21]. The first one is the computer configuration which includes the

device capacity and speed. The second one is the corpus properties such as language and input size. And finally we have the user profile [21].

Fig A.11: The DSM of the Smart Social project

Figure A.12: DSM of Smart Education

The figures A.11 and A.12 represent the Smart Social and Smart Education DSM respectively. The environment parameters are not represented in those matrices. The higher module, which is divided into three component parts, represents the design

rules module. It contains the system design rules, the platform design rules and the project design rules respectively. Hence, the system and platform design rules are the same for the two projects.

Table 1 provides the technical potential σ and the visibility of each module in the Smart Social project. We adopt those parameters to calculate the NOV.

Module Name	sigma	Z	Parameters per module
Design Rules	0	1	26
AuthenticationModule	2.5	0	27
CitizenPromotionModule	2.5	0	9
TraderPromotionModule	2.5	0	6
SupervisorPromotionModule	2.5	0	7
PublicityModule	1.6	0	5
CitizenPublicityModule	1.6	0	3
ServiceProviderPublicityModule	1.6	0	5
SupervisorPublicityModule	1.6	0	7
CitizenTransactionModule	2.5	0	8
ServiceTransactionModule	2.5	0	5

NotificationModule	2.5	0	3
EmailModule	2.5	0	3
AdministrationModule	2.5	0	9
QRcodeModule	1.6	0	5
ServiceProviderModule	2.5	0	2
AnnonceModule	2.5	0	8

Table1: Assumed Technical Potential and Visibility of Smart Social project

To compare the software system regulation added value with the software system engineering once, we take the same DSM for the engineering as the regulation and eliminate the design rules that we have added to cover much more higher levels. By starting with the smart social project, we used only **23** design rules for the engineering. As a result, the added value with the traditional engineering design reached **5.14** which is improved to reach **5.17** using our framework. The figure A.13 presents the corresponding plots of each module separately of the option values. The blue and the red colors represent our regulation framework and traditional engineering respectively. The option value of each module is the value at the peak.

Figure A.13: option value per module of the smart city system (contain only the smart social project)

To add new projects to the system, the design rules must be adopted to ensure the interoperability between the different services. As a second project, we chose to work on the Smart Education project. For Smart Education, the technical potential and the visibility of each module are provided in table 2.

Module Name	sigma	Z	Parameters per module
Design Rules	0	1	28
AdministrationModule	1.6	0	10
UniversityControlModule	1.6	0	7

SubmissionCandidateModule	1.6	0	11
SubmitCandidateModule	2.5	0	8
submissionControlModule	2.5	0	9
UniversityMobilitiesModule	2.5	0	8
StaffStudentMobilitiesModule	2.5	0	7
AuthenticationModule	1.6	0	33
EmailModule	2.5	0	9
NotificationModule	1.6	0	3
StaffAcademicModule	2.5	0	4
StaffAdministratifModule	2.5	0	4
StudentModule	2.5	0	4
SupervisorModule	2.5	0	4
QRcodeModule	2.5	0	4
CandidateHistoricModule	1.6	0	4

Table 2: Assumed Technical Potential and Visibility of smart education project

By using the traditional software engineering methodologies, each project is designed separately, so each project has its own design rules. Therefore, the engineering design rules contain the design rules of the first project as well the second one. By the way, we used **25** design rules for smart education engineering. So, to determine the system's

added value we need to concatenate the design rules for the engineering of all the designed projects to obtain **48** design rules. However, by adopting the regulation, the design rules that were already created in the first project must be adopted by the second one, so there is no need to redefine it each time. Therefore, the regulation reduces the number of design rules to **39**.

As a result of NOV analysis of the entire Smart City System, we got for the traditional engineering **8.77** which is improved with our framework to reach **9.26**. The figure A.14 represents the comparative plots of the option value for each module separately.

Figure A.14: option value of the entire system (after the smart education design)

As a result , our framework ensures the improvement of the system's added value with each new project design. The more new projects are designed, the more the added value is improved. This hierarchical design rules helps to create representative

architecture using the DSM and facilitate new project's design by constraining the design space exploration.

5. Conclusion and future trends

In this paper, we developed the FARUL3S to deal with ULSS systems management and development processes. The FARUL3S is a user-centered solution that aims at combining the complex adaptive system, the financial economics as well as the engineering systems design. We began this paper with the related works to introduce the fundamental concepts behind our Framework. Thereafter, we proceed to our solution so we provide the architecture and the detailed description of our framework FARUL3S. In fact, we don't focus on the mathematical foundations as we aim to illustrate the entire Framework steps.

The FARUL3S aims at creating a representative architecture and constraining the system component parts which make our framework a bottom-up solution. The entire system's added value is affected by the number of projects already developed. The more new projects are designed, the more the added value is improved.

As future trends, we propose to proceed to the automation of the management and design stages of the FARUL3S as it is crucial to reduce the time from hours to seconds. In this section, we aim to reduce the way for the researchers that want to

focus more on this trend. Thus, we provide some insights about the automation of the management and design phases based on our different lectures.

Starting with the map and validation of assumptions step. To improve the assumption's validation, we need a more reliable method based on mathematical approaches. Due to the effect of modularity among assumptions, such as reducing the cost change and decreasing portfolio complexity, it is indispensable to determine the optimal modularity degree throughout a set of ideas. Based on the DSM, Khastehdel and Mansour have developed a dynamic model for optimal idea selection during fuzzy front end [39]. The model exploits the idea's level of uncertainty and the dependency within the other ideas [39] so it can be a solution for the assumption's validation for a service as well for a platform through the design space's exploration.

For the design step, we found the Simon Tool [23, 35] that supports the design modeling and analysis framework. Furthermore, there is the Minos Tool [25] as a refactored version of Simon, it supports ACN modeling and a set of modularity analysis, such as the automatic derivation of the DSM, and Titan Tool which is a DSM viewer [26, 36]. Wong et al. present the DRH algorithm to automate the derivation of the DRH based on the condensation graph and a set of sub-ACNs [25]. Moreover, Cai et al. propose a set of algorithms for ArchDRH clustering and decomposing [26].

As multiple competitive and cooperative organizations can work on the same service at the same time, we need to identify the best design to be the start point of the next round. The AI represents an efficient technology to determine the optimal sequential design decisions. Tang et al. [40] present an optimization method to select the suitable component for a given function by performing an Ant Colony Optimization method. This

approach can be a solution for optimal design selection through a set of design-decisions made by different organizations.

For product development, an optimal design is obtained by ordering the design parameters with the goal of minimizing the DSM feedback and iteration [41].

Multiple intelligent solutions appear to automate the design process.

For example, to optimize sequencing design decisions, Meier et al. propose a competent genetic algorithm solution-based [41] and Shajahan et al. [42] contribute a Particle Swarm Optimization approach. On the other hand, Bonjour et al. propose a fuzzy method to produce the product architecture based on the functional one [43].

In fact, all those approaches are dedicated to the product's design and do not meet the software design requirements. While, it can lead to new insights in how to invest in AI to improve software design and provide more guidance for design spaces exploration.

As there are multiple competitive and cooperative organizations that may work on the same or different projects under the same system, keeping the system DRH up to date is critical as it provides incentives to create valuable designs. The system DRH must take into account the most successful design selection, so it must be adopted as the start point of the next round. Furthermore, to provide a real time notification about each new system DRH update, the publish subscribe pattern solutions such as the Data Distribution Service (DDS) can be a solution. In the DDS, once there are writers that publish data into a specific topic, the readers that were already subscribed to this topic receive notification about the new added data with no information about the writer.

6. ACKNOWLEDGMENT:

We deeply acknowledge Taif University for Supporting this study through Taif University Researchers Supporting Project number (TURSP-2020/327), Taif University, Taif, Saudi Arabia.

The research leading to these results has received funding from the Ministry of Higher Education and Scientific Research of Tunisia under the grant agreement number LR11ES48.

7. References

1. Northrop, L., Feiler, P., Gabriel, R. P., Goodenough, J., Linger, R., Longstaff, T., Wallnau, K., Ultra-large-scale systems: The software challenge of the future, Carnegie-Mellon Univ. Pittsburgh Pa Software Engineering Inst.(2006).
2. Evans, Eric. Domain Driven Design. In press, 2003.
3. Su, K., Li, J., Fu, H., Smart city and the applications, IEEE International Conference on Electronics, Communications and Control (ICECC), IEEE, Ningbo, China, 1028-1031 (2011).
4. Dell'Era, C., Landoni, P., Living lab: A methodology between user-centred design and participatory design, Creativity and Innovation Management, 23(2), 137–154 (2014).
5. Westerlund, M. Leminen, S., The Multiplicity of Research on Innovation through Living Labs. In Proceedings of the XXV ISPIM Conference, Dublin, Ireland, June 8–11 (2014).

6. Leminen, S., Living Labs as Open Innovation Networks - Networks, Roles, and Innovation Outcomes, Doctoral dissertation, Helsinki, Finland: Aalto University (2015).
7. Leminen, S., Westerlund, M., A Framework for Understanding the Different Research Avenues of Living Labs, *International Journal of Technology Marketing*, 11(4), 399–420 (2016).
8. Leminen, S., Westerlund, M., Nyström, A. G., Living labs as open-innovation networks, *Technology Innovation Management Review*, 2(9),6–11 (2012)
9. Leminen, S., Westerlund, M., Categorization of Innovation Tools in Living Labs, *Technology Innovation Management Review*, 7(1), 15–25 (2017).
10. Bergvall-Kåreborn, B., Holst, M., Ståhlbröst, A., Concept Design with a Living Lab Approach, *Proceedings of the 42nd Annual Hawaii International Conference on System Sciences*, HICSS '09, IEEE, 1–10 (2009).
11. Tang, T., Hämäläinen, M., Beyond Open Innovation: The Living Lab Way of ICT Innovation, *Interdisciplinary Studies Journal*, 3(4), 15–23 (2014).
12. Bergvall-Kåreborn, B. Ståhlbröst, A., Living Lab: an open and citizen-centric approach for innovation, *International Journal of Innovation and Regional Development*, 1(4), 356-370 (2009).
13. Cajander, Å., Larusdottir, M., Gulliksen, J., Existing but not explicit-The user perspective in scrum projects in practice, In: *Proceedings of the 14th IFIP TC 13 international conference human–computer interaction (INTERACT 2013)*. Berlin, Heidelberg. Springer,762-779 (2013).
14. Schwaber, K., *Agile project management with Scrum*. Redmond, Wash: Microsoft Press (2004)

15. Singh, M., U-SCRUM: An Agile Methodology for Promoting Usability,, in Proceedings of the Agile 2008, Toronto, 555-560 (2008).
16. Gothelf, J. Seiden, J., Lean UX: Applying Lean Principles to Improve User Experience, O'Reilly Media, Beijing ; Sebastopol, CA (2013).
17. Liikkanen, L., Kilpiö, H., Svan, L. Hiltunen, M., Lean UX - The next generation of user-centered agile?, pp. 1095-1100, Proceedings of the 8th Nordic Conference on Human-Computer Interaction (NordiCHI '14), ACM (2014).
18. Coenen, T., Robijt, S., Towards FALL: a Framework for Agile Living Lab projects, Proceedings of OpenLivingLab Days 2016, 129–144 (2016).
19. Coenen, T., Robijt, S.,Heading for a FALL: A Framework for Agile Living Lab Projects, Technology Innovation Management Review, 37-43 (2017).
20. Baldwin, C., K. Clark. 1999. Design Rules: The Power of Modularity. MIT Press, Cambridge, MA.
21. K. Sullivan, Y. Cai, B. Hallen, and W. G. Griswold. The structure and value of modularity in software design. SIGSOFT Software Engineering Notes, 26(5):99–108, Sept. 2001.
22. Parnas, D. L., “On the criteria to be used in decomposing system into modules,” Communications of the ACM, Vol. 15, No. 12, December 1972 pp. 1053–1058.
23. Y. Cai, “Modularity in design: Formal modeling and automated analysis,” Ph.D. dissertation, Dept. Comp. Sci., Univ. Virginia, Richmond, VA, USA, 2006.
24. Cai, Y., and Sullivan, K. J. 2006. "Modularity Analysis of Logical Design Models," in Proceedings of the 21st IEEE International Conference on Automated Software Engineering, September 18-22, Tokyo, Japan, pp. 91-102.

25. S. Wong, Y. Cai, G. Valetto, G. Simeonov, and K. Sethi. Design rule hierarchies and parallelism in software development tasks. In Proc. 24th ASE, pages 197–208, Nov. 2009.
26. Y. Cai, H. Wong, S. Wong, and L. Wang. Leveraging design rules to improve software architecture recovery. In Proc. 9th International ACM Sigsoft Conference on the Quality of Software Architectures, pages 133– 142, June 2013.
27. B. W. Boehm. Software engineering economics. In IEEE Transactions on Software Engineering (SE), volume 10, Jan 1984.
28. Sullivan, K., Chalasani, P., and Jha, S. "Software Design Decisions as Real Options," Technical Report, Carnegie Mellon University, Computer Science Department, June 1997.
29. Chalasani, P, Jha, S., and Sullivan, K. "An Options Approach to Software Prototyping," Working Paper, College of Computer Science, Carnegie-Mellon University, February 1997.
30. K. Sullivan, P. Chalasani, S. Jha, and V. Sazawal. Software Design as an Investment Activity: A Real Options Perspective in Real Options and Business Strategy: Applications to Decision Making. Risk Books, November 1999.
31. B. Boehm and K.J. Sullivan, "Software economics: a roadmap," in The Future of Software Engineering, 22nd International Conference on Software Engineering, June, 2000. pp. 319–344.
32. B. Boehm and K. Sullivan, "Software Economics: Status and Prospects," Information and Software Technology 41, 1999, pp. 937-946.

33. Abbas, H.; Shaheen, S.; Elhoseny, M.; Singh, A.K.; Alkhambashi, M. Systems thinking for developing sustainable complex smart cities based on self-regulated agent systems and fog computing. *Sustain. Comput. Inform. Syst.* 2018, 19, 204–213.
34. Schuurman, D., A-L. Herregodts, A. Georges, O. Rits, Innovation management in Living Labs Projects: The Innovatrix framework, Research and Innovation Conference Proceed-ings Open Living Labs days, ENOLL, 26-43(2018).
35. Y. Cai and K. J. Sullivan, “Simon: modeling and analysis of design space structures,” in Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering, ser. ASE '05. New York, NY, USA: ACM, 2005, pp. 329–332. [Online]. Available: <http://doi.acm.org/10.1145/1101908.110196>
36. S. Wong. On the Interplay of Architecture and Collaboration on Software Evolution and Maintenance. PhD thesis, Drexel University, Dec. 2010.
37. Smith, R. P., S. D. Eppinger. 1997. A predictive model of sequential iteration in engineering design. *Management Sci.* 43(8) 1104-1120.
38. Tian, Q. H. , Zhang, C. , Zhu, W. , Huang, J. K. , Zhang, Y. R., Zhou, X. M., 2020. Study on fuzzy duration of serial coupled iterative design tasks in product development.In IOP Conf. Ser.: Mater. Sci. Eng. 892 012080
39. M. Khastehdel and S. Mansour, “Developing a dynamic model for idea selection during fuzzy front end of innovation,” *Int. J. Econ. Manage. Eng.*, vol. 5, no. 3, 2018.
40. Tang D., Yin L., Ullah I. (2018) Matrix-Based Computational Concept Design with Ant Colony Optimization. In: Matrix-based Product Design and Change Management. Springer, Singapore. https://doi.org/10.1007/978-981-10-5077-0_4

41. C. Meier, A. A. Yassine, and T. R. Browning, "Design process sequencing with competent genetic algorithms," *J. Mech. Des.*, vol. 129, pp. 566– 585, 2007.
42. Shajahan, C., Subramanyan, E. E., Kumanan, S., and Firoz, N. (2019). Design process sequencing in product development process using design structure matrix and particle swarm optimisation techniques. In *J. of Physics: Conf. Series*, volume 1355, page 012027. IOP Publishing.
43. E. Bonjour, S. Deniaud, M. Dulmet, and G. Harmel, "A fuzzy method for propagating functional architecture constraints to physical architecture," *Trans. ASME, J. Mech. Des*, vol. 131, no. 6, pp. 061002-1–061002-11, Jun. 2009.