

Efficient software design paradigms for safety-critical systems (November 2015)

M.Sc. Kurosh A. Farsi Madan
 Degree student in Computer Science
 School of Business and Management
 CT60A5101 Models and Methods of Software Engineering
 Lappeenranta University of Technology
 kurosh.farsi.madan@student.lut.fi

Abstract— Future software systems will be more automated and far more dependent on safety-critical computers than the present day software systems even though our current systems are highly advanced [5] [36].

The software systems will be highly complex, integrated, large, and distributed which makes them complicated for most of the humans to grasp and manage. Furthermore they are device agnostic which makes them highly prone to integration issues [5].

There are literally millions of different variables and conditions that could go wrong since their functions are orchestrated between various software independent and managed systems. They are safety-critical or they somehow work on safety-critical systems, which means that in order to meet the stakeholder requirements, they might need the same or similar type of validation procedure. Even a simple iPhone app can be a safety-critical software depending on different factors [5] [36].

A safety critical system is a software that is considered safety-critical if it controls or monitors hazardous or safety-critical hardware or software and the characteristics of safety-critical systems are the following where the system controls hazardous or safety-critical hardware or software, monitors safety-critical hardware or software as part of a hazard control, provides information upon which a safety-related decision is made, performs analysis that impacts automatic or manual hazardous operations, verifies hardware or software hazard controls, can prevent safety-critical hardware or software from functioning properly.

Examples of safety-critical systems are well-known unfortunately, only when failures happen. Known examples are to be found on various fields for example in the medical devices, rockets, business applications, airplanes, and automobiles where a faulty or buggy software caused a hazardous event where the safety of people were in danger or there was a loss of life [36] [11] [10].

The reasons for system failures differ from case to case but the hazards were usually the result of a bad software design to handle the faults or system errors not to mention that the software itself was the source of the fault and hazardous event because of a buggy feature in the calculations.

The date of submission was 20.11.2015. The research paper was written with no sponsors. The work was partially supported by Lappeenranta University of Technology with free access and usage of books, articles, research papers among other various sources of information.

This paper cannot go through, every field of engineering to trawl and find the most proper paradigm in designing a safety-critical system since to design a proper architecture for a system is even a real challenge to many large organization and companies let alone for a particular individual so the focus is on finding the proper approach in designing a safety-critical system in the field of medical science for an insulin pump in theoretical level.

Index Terms— paradigm, methodology, safety-critical system, Object-Oriented, formal methods.

I. INTRODUCTION

THIS paper seeks to find a compatible and efficient approach in designing a medical device where the case study is an Insulin Pump which is a safety-critical system. Note that the author of this paper did not write ‘the best approach’ since there are numerous paradigms, methodologies, tools, and techniques in designing and implementing the system architectures in the different fields of engineering and every approach has its benefits without forgetting the fact that the companies that design and sell successful and unsuccessful devices or device prototypes, usually do not publish the correct procedures and design methods to the public since it could damage the company reputation if some process phase was erroneous.

Whatever the reasons were in the failures and successes, most of the safety-critical methodologies used today were originally developed for more hardware and electro-mechanical oriented systems whereas our devices are becoming more software oriented since it allows more complexity in the system being developed [4] [36].

Medical devices for example, rely more on the software than the hardware these days since they execute complicated calculations and procedures impossible for a manual human execution. They are devices that we expect to heal and help us in the times of urgency related to our health and we expect them to work for our benefit. A medical device should never harm

Kurosh A. Farsi Madan is a M.Sc. student in Computer Science, Lappeenranta University of Technology, Skinnarilankatu 34, 53850, Finland (e-mail: kurosh.farsi.madan@student.lut.fi).

and injure a human being and yet we have heard cases where they have done so either through a buggy functionality or missing fault controls thus making them safety-critical systems.

The sources of safety hazards range from radiology devices to heart implants and causes for most of the faults are related to a certain and specific functionality of the whole device where a formal validations and verifications might have been implemented on a very shallow level, invalid way, or not at all. Another usual reasons for these failures stem from the information flow and interaction design between different systems components [33].

In the near future when the medical devices are more than firmware and hardware where the device is heavily software oriented; we have to take safety of the device even more in consideration since they could be integrated to various other systems for example real-time updatable databases, other devices, and so on. They should be free from unacceptable risks and avoid physical injury and damage to health of people by all means [5] [36].

To be able to ensure the safety of future medical devices, there is a need to incorporate various methods to trawl through the system functionalities from top to bottom. The methods include government standards (FDA¹), guidelines (IEC²/ISO³), software specification requirements, proper architecture design and implementation tools, life cycle and development methodologies (Agile methods). For example an infusion pump for insulin pump is considered as a safety-critical software-intensive medical device (SMD), on which FDA has specified recommended regulatory requirements that demonstrates that the device is safe and effective although it does not take a stance on what particular software development methodology should be used [28] [25] [33] [40].

This paper will mainly focus on finding the proper software design and validation paradigm and tools. Furthermore the paper will go through briefly, one of the mostly used software development methodologies like SCRUM, its structure, and what benefits it might bring in the development phase itself or if it should be used at all with other similar agile methodologies.

To know what paradigms and tools to use, I will have to find, inspect, analyze, compare and combine the SCS characteristics like how large and integrated the system is, what types of functionalities it has and what kinds of measurements the system will have on the safety, reliability, availability, maintainability, integrity, and confidentiality attributes and what metrics to use in the first place. What kind of failures, faults, and errors might arise and what might be the causes, situations, and the expected outcomes. What about the unknown situations?

I can only design and validate the known issues but I cannot do a formal validation on the unexpected behavior that was not taken into consideration. There is only a limited amount of different ways to make a software and hardware (SCS) fault tolerant and it to be able to remove the fault and maybe even forecast the faults to prevent safety hazards. Even expected faults might not have beneficial fail-stop behavior let alone the

unexpected ones to ensure the safety of living beings or property or at least minimize the damage.

II. COMPONENT BASED DEVELOPMENT

The first modelling and design approach topic that I will go through is component based modeling (CBD).

CBD is an approach where we design a system from prebuilt software components or we design our system to have reusable components that can be used in other parts of the system or in another project [19] [59].

Every component is composed of functionalities related to each other instead of designing one large software system.

The benefits of CBD usage are the same as in object-oriented approach i.e. reusability, reduced production costs, shorter development time, better quality control among other [59].

CBD complements object-oriented design (which I will cover in the next chapter), since our modern systems are too large and complicated to just create the architecture as one big set of objects since some objects should have one set of functionality than the others and being able to design the systems as different system component parts will help to maintain the software and move the different component parts or integrate more components if needed. This is widely known as n-tier structure. (MVC for example) [20] [34] [35] [39]. According to Bose, the component-based software engineering (CBSE) the following [20]:

“A branch of software engineering which emphasizes the separation of concerns in respect of the wide-ranging functionality available throughout a given software system. This practice aims to bring about an equally wide-ranging degree of benefits in both the short-term and the long-term for the software itself and for organizations that sponsor such software... It is a nontrivial, nearly independent, and replaceable part of a system that fulfills a clear function in the context of a well-defined architecture.”

If the system is built by using CBD approach, then the components should in theory be available without any restriction in other programming languages and hardware's through integration i.e. one unit of component should be easily reusable in other software system by using interfaces [20].

The benefit again is that once the component has been designed and implemented; the safety validation made for the component objects should be still valid, even though FM type of validation and testing should always be present whenever there is any kind of changes made to the design and structure of an architecture. Below I have tried to showcase different system components with their own sets of classes:

¹ FDA – Food and Drug Administration

² IEC - International Electrotechnical Commission, that publishes standards and conformity assessment systems to perform, fit and work safely together.

² ISO – International Organization for Standardization.

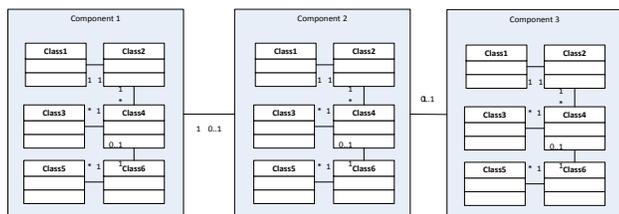


Figure 1: System components

The usage of components is not always necessary but since our case study is a safety-critical system and the device is an Insulin Pump that contains hardware, software, firmware, and removable parts; it is critical to have the CBD since components are on some cases hardware dependent and some functionalities can be duplicated for future device models.

The benefits that I am seeking are not in the traditional sense the following where only abstraction, data and information hiding, and functional independence are usually sought after. I believe that CBD will help in developing any kind safety-critical system by segmenting and dividing the system and hardware functionalities into their own set of components where the separation of logic will help in maintaining the hardware, objects, and functionalities.

I will have to emphasize that CBD has its own set of processes and guidelines for designing the components but they will be out of scope of this research paper.

III. OBJECT-ORIENTED DESIGN

There is a lot to be said about using object-oriented approach to developing safety-critical systems. There is more than enough quantitative and qualitative data available for analyzation and reference both for the good and bad and the arguments for both of them are based on solid statistics and grounds [4] [1].

The only problem is that there is still no clear ground on the reasons for either argument since the usage of OO approach for developing safety-critical systems has been more or less on using OO approach as a standalone paradigm combined with few tools from other methodologies. Rarely do we see software designs to use object-oriented approach alone, independent on the type of system that is being developed. For example by just using OO in the design, there might be safety issues since classes and the description of their attributes, functionalities and other relationships do not really explain on how the system should or is supposed to work even if there is a heavy software requirement documentation available with a more detailed criteria points. Furthermore, the de-facto tool used to model OO called UML will hide some functionality since there is a limitation on how much you can simulate the real-world by using the OO concepts.

In this paper, I will try to search of the beneficial aspects in merging the missing characteristics of OO with Formal Methods to overcome the safety and other issues that might arise in designing a safety-critical system by first going through some general beneficial aspects of OO and then how it might be used in designing SCS's.

Studies [3] have shown that teams that use OO usually consume less time in the maintenance phase of the software system even though they usually generate a lot of code, objects,

and components. Furthermore, the generally known benefits of using OO with UML are encapsulation, high cohesion, and low coupling through modelling of the requirements and domain properties of the application domain not to mention that it can be implemented with almost any programming language and technology [1] [2].

The less time is spent on fixing the code, the lesser the expenses are. This is because UML (a general modelling language used for OO without any specific standardization) provides good analysis tools for comprehension and communication of abstract ideas to the development team by providing a complete picture to the development team and thus relieves the 'mental load' on them but as stated previously in the paper; the UML in itself does not have safety and security checks and it was adopted as one of the Object-Oriented design tools [2].

The writer of this paper; even though strongly tried to fight against the urge to bring out his own opinions, agree with the stated information.

Even though the number of components or files do increase in the project; everything is divided and segmented in to different logical and understandable components. Object-Oriented basically provides:

1) Code reusability which means that for example the previous project that the team was working on, might have usable components and classes for the new and existing project. The core of reusability of classes and components are the design pattern/s used in them by giving characteristics like inheritance, possibility for extension etc. which are separate topics in themselves [1] [2] [3].

2) The whole principle of OO is based on "objects" or "classes" by using the software terms. Objects are models of real world. Objects can represent any type of real world physical things and concept that might help us to understand, what happens in our application. An object named Person might encapsulate all information that a real person might have like first name, last name, age, social security number etc. The object might also resemble a physical object like a sensor, engine etc. This will help to divide the objects into components that have the same set of functionality. [30]

Another example could be the following where you might have a car which is the software system itself. You have different car engines and physical parts with controlled software (not firmware). The software has been divided in to different sets of components and components into classes. All of this can be encapsulated in the OO diagram. The hull of the engines or the car can be represented as different objects or components. What is going on inside is like viewing a black box for us. We do not know what functionality is and the attributes this black box or boxes have.

3) Object-Oriented combined with the UML are universal paradigms and methodologies that contain design patterns and terminologies that are transferable independent of any technologies and platforms like in the following list [3] [1]:

- Classes
- Instances
- Nodes

- Inheritance
- Actors
- Operators / events
- Operator overloading (method overloading)
- Type safety
- Encapsulation
- Abstraction of real world objects
- Aggregation
- Association
- Cohesion
- Collaboration
- Coupling
- Modularity
- Hierarchy
- Attributes
- Functions
- Relationships (connections/interactions between the designed objects)
- Inheritance
- Instances
- Polymorphism (Toyota → Car → Vehicle (extends))
- Dynamic binding
- Constructors/destructors
- Getters/setters and so on

So to recap objects or in other words classes; they are reusable, extendible, and modular i.e. once an object have been created; it can be instantiated in any part of the system. It doesn't need to be coded again in different parts and when you need to extend an object; you can do so inside one class for example if the Person class should contain the social security number attribute and getter/setter methods, you can implement the new attributes inside one object (available everywhere). If there are other types of 'Persons' for example disabled people with their own set of attributes for some reason. The changes made in one place are available for every instantiated Person object in the extended classes.

The designed and implemented OO software is much easier to maintain since every class has encapsulated one business aspect or property. For example a change in the some system functionality; should not need a lot of code rework since all real world "Objects" or concepts have their corresponding class. A car, building, animal, passport, bill/facture, or anything else for that matter; can be encapsulated inside classes and classes can be encapsulated and segmented in different system components and so on.

Furthermore, the reader should also note that even though IEEE [7] has stated and defined what the maintenance of the software is after the implementation phase; the same definition also can be applied to the design phase with maintainability in mind. The following definitions are valid in maintenance and maintainability design and rework phase:

"Modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment."

And

"The set of activities that takes place to ensure that software installed for operational use continues to perform as intended and fulfill its intended role in system operation. Software maintenance includes improvements, aid to users, and related activities."

By taking the previous statement into consideration; now try to imagine a linearly designed (or not designed) system architecture with all the controlling (controllers), viewable parts (views), objects (representation of business logic and real world physical objects and concepts), and functionality related codes are mashed and designed into only few classes (no logical and clear separation of codes). If the design has to have formal validation and the code is not coherent then the formal validation will not produce any reliable results since all the actors and system behavior cannot be predicted or even calculated, since the amount of variables (not system variables like integers, chars, and strings) will rise in the validation phase. The difference between the linear or in other words procedural types of programming styles with Object-Oriented is that the non-object oriented ways to design and develop software systems are called spaghetti code or in other words coding from top to bottom. OO will provide us necessary information to evaluate if the designed objects meet the requirements and redesign different objects based on use cases [34].

Basically OO design is all about diving the system into classes and objects to model the real world behavior where functionality is based and coordinated between different classes through instantiation and other techniques.

There is again the possibility to inject all the functions / methods into one class where everything will be handled but then all the benefits of OO design would disappear and it would not be OO design anymore. Furthermore, as stated earlier, UML can provide the benefit of (complement) designing OO architecture without losing the benefits of OO but being able to design the system in a way that is functionally safe by implementing the functional requirements on different classes where needed.

UML will also literally provide a whole picture of the whole system design so the safety requirements can be allocated in different classes and validated in a way that we have a thorough understanding of the program process flow.

Below I have tried to draw a very plain UML class diagram where the point is to showcase a controller class related to specific Insulin Pump component and functionality designed by using OO design. Although creating classes that instantiate objects do not mean that the architecture conforms to OO principles; the reader can assume that it uses OO guidelines:

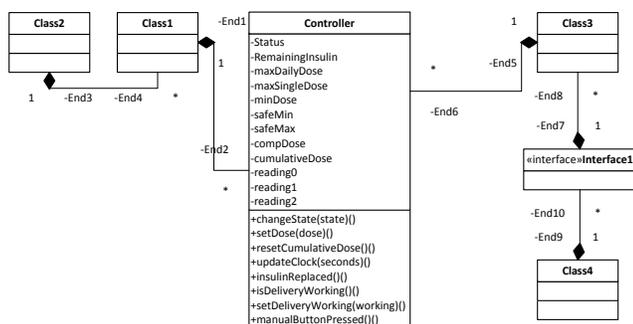


Figure 2: Insulin Pump component class diagram

In the above diagram the flow can be seen as relationships and the design can help the architect to further develop more safer segmented OO design for the different critical functionalities for example bolus and basal calculations in their own sets of classes where validation can be easy since the relationships in itself will show the data flow between the classes.

There are of course other tools in the object-oriented UML like flowcharts, fault trees, analysis state charts, object constraint language that is a tool for UML, use case diagrams, and so on where some should be created before another but since the paper would become longer than a book, and I will focus only on OO UML class diagrams. The reader can assume that fault tree was used to find the most critical functionality that needs further validation and verification and the reason is that the logical gates in FTA will help us to write formal validations more precisely because OO UML charts, diagrams, and descriptions are not good enough for safety checking the data value aspects.

So to summarize it all; it is hard to analyze function failure in OO system because functionally decomposed systems are easier to be analyzed since there is the possibility to pinpoint the functional failure in a system.

IV. FORMAL METHODS

Formal methods is a technique used widely in different domains where problem solving and validation is necessary i.e. we want the correctness of the program to be verified by proofs [24]. Fields where formal methods can be used range from social, marine, business, engineering, to medical domains etc.

Formal methods [12] are a way to transfer ideas or informal text, words and pictures into formal space where it is in a form where it can be modeled and the technologies can be chosen, applied and implemented for it since the stated informal specifications are ambiguous and imprecise.

Formal methods are basically mathematics oriented approaches for designing / implementing error free software and hardware systems by using specifications, verification / validation / proofing of hardware and software systems etc.

The expectation in the use of formal methods are reliability and robustness of the Software. Sommerville [20] defined formal methods as following:

“The term ‘formal methods’ is used to refer to any activities that rely on mathematical representations of software including formal system specification, specification analysis and proof, transformational development, and program verification.”

The level of mathematics will vary but usually it is based on logic and algebraic notation so it can be considered in every case to have a sound basis in mathematics. Formal methods provide a way to inspect the system functionality [24] where by using OO, CBD, and UML you can mostly inspect the system structure.

The approach was introduced in the 1930’s by a collection of people from different fields to describe the problems in an understandable way so that it can be solved. It is not a technical specification of the implementation but a guideline for validating, developing, and designing the software system. The word formal means [13] a computer executable logical solutions of a problem i.e. it can be machine checked. By using formal methods you are basically doing a representation of your problem, the calculation for fixing the problem and the expected solution / output for the problem.

Formal method is a good tool and paradigm for checking the functional and data value aspects of the developed software system or in our case the Insulin Pump.

It will helps us to design and validate the different safety-critical functions that are calculating and modifying data / values intensively where a small miscalculation could lead to a loss of life. Since object-oriented UML tools hide the data crunching aspects of the design; the formal methods basically complement the OO paradigm.

FM also provides a good foundation for verifying large and complex systems. The paradigm provides a logical and grounded reasoning of the system functionalities and provides a solid foundation to the system development.

For example, take into consideration the failure in the insulin delivery with the pump infusing too high dose of insulin due to overflow or air pressure in lines. Without having a good set of pre-designed validations and verifications, the implemented solution might miss this probable risk. FM provides a good set of tools for fault checking the dose calculation algorithm and validation of the requirements for the dose delivery [31].

So to summarize formal methods, the benefits of using formal methods are validated system functionalities, reusable specification for the systems whole life cycle (constant testing and validation available) and drawbacks are the scarce population of mathematicians who have some understanding in CS and that the method would consume too much money, resources and time if it would be implemented on a whole software systems for example below is a small FM example [20] from Sommerville’s book called “Software Engineering”. The validation is for an Insulin Pump where the device is in a run state with a certain run conditions (case):

```

//Input device definition
switch?: (off, manual, auto)
ManualDeliveryButton?: {}
Reading?: N
HardwareTest?: (OK, batterylow, pumpfail, sensorfail, deliveryfail)
InsulinReservoir?: (present, notpresent)
Needle?: (present, notpresent)
clock?: TIME

//Output device definition
alarm?: (on, off)
display1?: string
display2?: string
clock?: TIME
dose?: {}

// State variables used for dose computation
status: (running, warning, error)
r0, r1, r2: N
capacity, insulin_available : {}
max_daily_dose, max_single_dose, minimum_dose: {}
safemin, safemax: {}
CompDose, cumulative_dose: {}

r2 = Reading?
dose! ≤ insulin_available
insulin_available ≤ capacity

// The cumulative dose of insulin delivered is set to zero once every 24 hours
clock? = 000000 ⇒ cumulative_dose = 0

// If the cumulative dose exceeds the limit then operation is suspended
cumulative_dose ≥ max_daily_dose ∧ status = error ∧
display1! = "Daily dose exceeded"

// Pump configuration parameters
capacity = 100 ∧ safemin = 6 ∧ safemax = 14
max_daily_dose = 25 ∧ max_single_dose = 4 ∧ minimum_dose = 1

display2! = nat_to_string(dose!)
clock! = clock?

```

Figure 3: Insulin Pump Schema

In the above example Sommerville has specified a schema for an Insulin Pump through using formal methods.

```

AINSULIN_PUMP_STATE

switch? = auto
status = running ∨ status = warning
insulin_available ≥ max_single_dose
cumulative_dose < max_daily_dose

// The dose of insulin is computed depending on the blood sugar level
(SUGAR_LOW ∪ SUGAR_OK ∨ SUGAR_HIGH)
// 1. If the computed insulin dose is zero, don't deliver any insulin
CompDose = 0 ⇒ dose! = 0
∨

// 2. The maximum daily dose would be exceeded if the computed dose
was delivered so the insulin dose is set to the difference between the
maximum allowed daily dose and the cumulative dose delivered so far
CompDose + cumulative_dose > max_daily_dose ⇒ alarm! = on ∧
status' = warning ∧ dose! = max_daily_dose - cumulative_dose
∨

// 3. The normal situation. If maximum single dose is not exceeded
then deliver the computed dose. If the single dose computed is too high,
restrict the dose delivered to the maximum single dose
CompDose + cumulative_dose < max_daily_dose ⇒
(CompDose ≤ max_single_dose ⇒ dose! = CompDose
∨
CompDose > max_single_dose ⇒ dose! = max_single_dose)
insulin_available' = insulin_available - dose!
cumulative_dose' = cumulative_dose + dose!

insulin_available ≤ max_single_dose * 4 ⇒ status' = warning ∧
display1! = "Insulin low"

r1' = r2
r0' = r1

```

Figure 4: Insulin Pump Schema in a run state

In the above example Sommerville has specified a run state with a condition for the Insulin Pump. Notice that the mathematician or the person who is doing the validation can and will create many case / run conditions to validate all the safety critical functionalities and process flows.

By using formal methods, we can validate whatever we want to be validated for example by using the Z notation, we are able to create a validation for the state chart (UML tool) if it is well formed or validate a system functionality from scratch and apply them to the UML diagrams and charts so that the process is an iterative and constant modification of different parts of the design.

After everything has been validated and the designs are checked. The software requirements specification is basically complete at this phase whether there existed an SRS document before the design or it was compiled during the design and validation phase. The next step is to implement and develop the software system by using the SRS, designed architecture, and this is usually the projects start point.

V. SCRUM METHODOLOGY

As a part of this paper I will see whether or not; the agile development methodologies are suitable in the design and implementation of the SCS's. Scrum is a software development framework [30], mainly used for complex systems (although not limited to) these days. Scrum is a methodology where a group of people or in our case the developing team can address the scope and problem of the project by segmenting it to productive chunks of manageable components and tasks.

The roots of the framework [26] stem from the 1990's when there was a need for a lightweight, fast, understandable, and manageable project coordination.

Scrum framework has Scrum teams, role based task assignment, events, artifacts, rules, and more [26].

The difference between an agile (Scrum) software development methodology and a waterfall model or V-Model types of methodologies are the following:

- 1) Before agile, the development happened by first gathering all the requirements of the customers, analyzing the problem, and designing the whole software system from top to bottom before implementing anything.
- 2) Process flow in a waterfall or similar type of methodology was to first gather all the software requirements, plan, design, test, implement, and then deploy the software.

The benefits of using Scrum are various. Few points are listed below [27]:

- 1) It allows the gathering of requirements specification to happen before or concurrently in iterative stages. This allows more organic work culture where the customer can take more part in the development of the application whether to give more details regarding a specific requirement or to change the software characteristics. This way, all of the customer business needs are met and the software lifecycle is much longer since a typical software project can take more than 6 to 9 months where the business requirements most certainly change.
- 2) The customers can have more understanding of the system being developed by having scheduled and continuous meetings with the developing team. This way, no errors can creep into the system and the changes after the implementation will not be costly for both the customer and the developing company.
- 3) The iterative development minimizes risks since the regular demonstration and meetings with the customers will help and change the done work/tasks if they are not accepted. The developing team will get the feedback and if the deployed work does not meet the customer requirements, then there will be additional requirements gathering for rejected task / functionality.

The reasons for considering the usage (or not) of SCRUM methodology in the design and development of a safety critical system are the following.

Today's development environments are hectic. The pressure to produce high quality software in minimum amounts of time with lower costs have become the main goals in the software lifecycle management. Thus an ever-growing number of software development teams and companies are turning to SCRUM and agile methodologies in most if not all of their

projects as a consequence of pressure to survive in the highly competitive market despite the risks that SCRUM methodology brings.

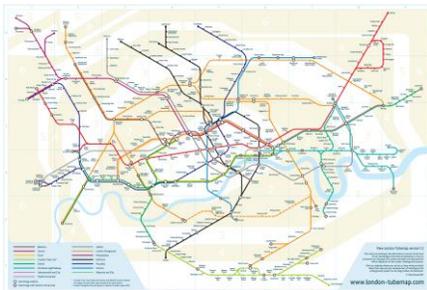
The drawbacks thus are the risks of going against the safety-critical software requirements assurance compliances. SCS have a very strict regulations, standards, and guidelines imposed on them and not every characteristic of SCRUM methodology follows these rules [35]. Furthermore, studies have shown that changes in the requirements can result in mistakes in the product/scrum Backlog and User Stories which could have had a detrimental impact on the end product. So we can conclude that SCRUM and agile methodologies in general are not suitable for safety-critical system development and design and the used method of designing and implementing the insulin pump or any other SCS is the plan driven or waterfall model type of software life cycle development.

VI. RESULTS

Using object-oriented approach to design the system architecture, will help us to segment the software into system components based on the system behavior and further divide the components into sets of classes with same kind of functionality where every class or object is imitating a real world property / object.

Object-orientation will allow us to use UML where the level of abstraction will be concise, complete, unambiguous, maintainable, comprehensible and cost-effective.

UML gives the designer and the programmer who implements the system, a good conceptual model. A good case example would be the underground subway system map [17] as shown below:



Picture 1: London tube map

Diagrams are basically tools to only show us the connections. Engineers and designers could hypothetically on a very shallow basis, conclude the most dangerous points in the underground tube / metro system and further investigate and analyze the safety requirements.

Object-orientation also allows us to further focus on validation of safety-critical functionalities or functions. By having a clear understanding of class relationships and different functions, we can easily apply formal methods on any function that needs validation.

Formal methods are the final step of the design phase before any kind of implementation. This is the most critical phase of the design since it will mathematically go through the system safety-critical functionality by proofing. The proofing will ensure that before implementing something; we are

absolutely sure that the wanted safety requirement will behave as we want it to. This way, no logical error will be able to creep into our system [16]. FM is basically the final frontier before starting the real work.

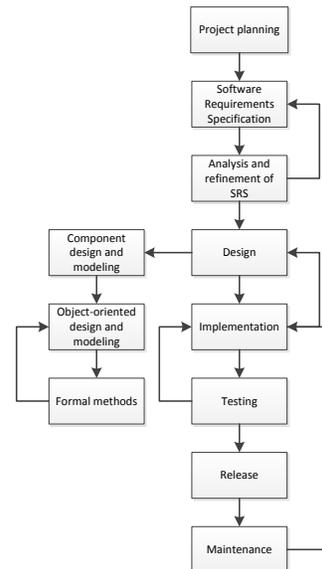


Figure 5: Plain and simple software life cycle

To implement the designed system, the first thing that comes in mind is that the requirements gathering and the design phase happened by using the waterfall methodology. This is true since a safety-critical system will need an engineer's approach but this does not rule out that the SRS and design can be thoroughly gathered and done before the implementation phase which is using the Scrum methodology. What the previous sentence meant was that the SRS and design can be complete and redefined while the project is being implemented. This takes more time but it is the most logical approach since safety-critical system development needs at least some kind of detailed overview so that the iteration development phases can be carefully designed by analyzing and prioritizing the system components.

Scrum methodology will bring the promised benefits listed in the Agile Manifesto like interaction, working software that meets the requirements, comprehensive customer collaboration, and the possibility for responding to any change in the plans. Just because we are using Scrum, does not mean that we cannot have a completely gathered software requirements specification, validated it, and designed the software architecture with the formal methods approach [38] but it is still highly recommended to not use this following methodology and to use plan driven or waterfall model instead.

So the conclusion is that by using Component Based Development for designing the hardware and software level combined with Object-Oriented (with UML) paradigm and Formal Methods in the design phase of an safety-critical system, will bring all the benefits of each paradigm needed for SCS without having any shortcomings of any kind since all of listed tools and techniques complement each other and fill some particular need for designing an insulin pump device. Furthermore, the suggested development methodology for the

software is either plan driven or sequential waterfall model since it would fit more appropriately for these types of design methodologies and tools.

REFERENCES

- [1] Certification Authorities Software Team (2000). *Object-Oriented Technology in Civil Aviation Projects*. Available: http://www.faa.gov/aircraft/air_cert/design_approvals/air_software/cast/cast_papers/media/cast-4.pdf
- [2] J. S. Lim & S. R. Jeong & M.C. Whang & Y. Cho. (2004). *Practical Investigation into the Maintainability of Object-Oriented Systems for Mission Critical Business*. The Graduate School of Business IT, Kookmin University. Seoul, Korea. Springer. Available: http://link.springer.com/chapter/10.1007/978-3-540-30198-1_56
- [3] S. Wambler. (2004). *The Object Primer*. Cambridge University Press. Available: <http://www.agiledata.org/essays/objectOrientation101.html>
- [4] R. Hawkins & I. Toyn & I. Bate. *An Approach to Designing Safety Critical Systems using the Unified Modelling Language*. Available: <https://www-users.cs.york.ac.uk/rhawkins/papers/finalUML.pdf>
- [5] Knight, J. C. (2002). *Safety Critical Systems: Challenges and Directions*. International Conference on Software Engineering, Orlando, FL. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1007998&tag=1
- [6] G. Stubbings & S. Polovina. *Levering object-oriented knowledge for service-oriented proficiency: Enhancing service capability in developers*. Available: <http://link.springer.com/article/10.1007%2F:s00607-013-0304-6>
- [7] IEEE. (2003) *IEEE Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations*. Available: <http://dictionary.ieee.org/index/s-14.html>
- [8] Dr. Homayoon, D. & Dr. Benjamin, A. & Dr. Everett, C. & Dr. Smith, C. & Dr. Stamatelatos, M. & Dr. Youngblood, R. (2011). *NASA System Safety Handbook: Volume I, System Safety Framework and Concepts for Implementation*. NASA. Available: <http://www.hq.nasa.gov/office/codeq/doctree/NASASP2010580.pdf>
- [9] O'Connor, B. (2004). *NASA Software Safety Guide*. NASA Available: <http://www.hq.nasa.gov/office/codeq/doctree/871913.pdf>
- [10] BBC. 1984: Hundreds die in Bhopal chemical accident. Available: http://news.bbc.co.uk/onthisday/hi/dates/stories/december/3/newsid_2698000/2698709.stm
- [11] Douglas, N. A. (2000). *The Explosion of Ariane 5*. University Of Minnesota. Available: <https://www.ima.umn.edu/~arnold/disasters/ariane.html>
- [12] Papadopoulos, Y. & McDermid, J.A. (1996). *A Harmonized Model for Safety Assessment and Certification of Safety-Critical Systems in the Transportation Industries*. Springer. Available: <http://springer.com/>
- [13] Gabbar, A.H. (2006). *Modern Formal Methods and Applications*. Springer. Available: <http://springer.com/>
- [14] Koopman, P. (2014). *Critical Systems and Software Safety*. Carnegie Mellon. Available: https://www.ece.cmu.edu/~ece649/lectures/19_critical_systems.pdf
- [15] Hawkins, R.D. (2006). *Using safety contracts in the Development of Safety Critical Object-Oriented Systems*. University of New York. Available: <https://www.cs.york.ac.uk/ftplib/reports/2007/YCST/04/YCST-2007-04.pdf>
- [16] Woodcock, J. & Davies, J. *Using Z: Specification, Refinement, and Proof*. University of Oxford. Available: <http://www.usingz.com/text/zedbook.pdf>
- [17] Pavlus, J. (2011). *London Tube Map Sparks Furor Over What "Design" means*. Available: <http://www.fastcodesign.com/1664692/london-tube-map-sparks-furor-over-what-design-means>
- [18] Kung-Liu Lau & Zheng Wang. (2007). *Software Component Models*. Available: http://search.proquest.com/docview/195576634/fulltextPDF?source=fed_srch&accountid=136582
- [19] Bose, D. *Component Based Development: Application in Software Engineering*. Indian Statistical Institute. Available: <http://arxiv.org/ftp/arxiv/papers/1011/1011.2163.pdf>
- [20] Sommerville, I. (2009). *Software Engineering*. Pearson. Available: https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/WebChapters/PDF/Ch_27_Formal_spec.pdf
- [21] *Hazard and Operability (HAZOP) studies*. University of Auckland. Available: <https://cdn.auckland.ac.nz/assets/ecm/documents/Hazard-Operability-Studies.pdf>
- [22] Croecker, D. *Safe Object-Oriented Software: The Verified Design-By-Contract Paradigm*. Available: <http://www.cs.toronto.edu/~hehner/FMSD/>
- [23] Hehner, E. *Formal Methods of Software Design (Online Course)*. University of Toronto. Available: http://www.eschertech.com/papers/safe_oo_software.pdf
- [24] Barroca, L. M. & McDermid, J. A. *Formal Methods: Use and Relevance for the Development of Safety Critical Systems*. Available: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.48.7150&rep=rep1&type=pdf>
- [25] Koopman, P (2014). *Chapter 19: Critical System and Software Safety*. Available: https://www.ece.cmu.edu/~ece649/lectures/19_critical_systems.pdf
- [26] Schwaber, K., & Sutherland, J. (2013). *The Definitive Guide to Scrum: The Rules of the Game*. Scrum Inc.
- [27] Blankenship, J. & Bussa, M. & Millet Scott (2011). *Pro Agile .NET Development with SCRUM*. Apress. (p 24 – 35)
- [28] Fowler, M. & Highsmith, J. (2001). *Agile Manifesto*. Available: <https://assets.uisi.edu/pdf/Agile-Manifesto.pdf>
- [29] Levenson, N. G. (2009). *The Need for New Paradigm in Safety Engineering*. Springer-Verlag London Limited. Available: <http://link.springer.com/book/10.1007/978-1-84882-349-5>
- [30] Easterbrook, S. (2005, 1 1). *Lecture 11: Object Oriented Modelling*. Retrieved 11 16, 2015, from University of Toronto. Available : <http://www.cs.toronto.edu/~sme/CSC340F/slides/11-objects.pdf>
- [31] Gorski, J., & Lukasiewicz, K. (2013). *Towards Agile Development of Critical Software*. *Software Engineering for Resilient Systems* (pp. 48-55). Ukraine: Springer.
- [32] Mashkoor, A., Biro, M., Dolgos, M., & Timar, P. (2015). *Refinement-Based Development of Software-Controlled Safety-Critical Active Medical Devices*. *7th International Conference* (pp. 120-132). Vienna, Austria: Springer. Retrieved from http://link.springer.com/chapter/10.1007/978-3-319-13251-8_8
- [33] Kaye, R., & Crowley, J. (200). *Medical Device Use-Safety: Incorporating Human Factors Engineering into Risk Management*. Washington D.C.: Springer.
- [34] Crnkovic, I., & Larsson, M. (2001). *Component-Based Software Engineering - New Paradigm of Software Development*. MIPRO. Västerås: Mälardalen University.
- [35] Leff, A., & Rayfield, J. T. (2001). *Web-Application Development Using the ModelNewlController Design Pattern*. Watson Research Center. Watson: IEEE.
- [36] Dale, C., & Anderson, T. (2009). *Safety-Critical Systems: Problems, Process and Practice*. Brighton: Springer. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6800301&queryText=safety-critical%20system&pageNumber=2&newsearch=true&searchField=Search_All
- [37] Dr. Setzer, A. (2005). *High Integrity Systems & Critical Systems*. University of Wales Swansea. Available: <http://www.cs.swan.ac.uk/~csetzer/lectures/critsys/04/critsysfinalforprinting0.pdf>
- [38] Dewar, R. B. K. *Safety-Critical Design Techniques for Secure and Reliable Systems*. *Embedded Systems Conference*. Available: http://www.adacore.com/uploads/technical-papers/Safety-Critical_Design.pdf
- [39] O'Brien, S., & Shmeltzer, S. (2011). *Oracle Application Development Framework Overview*. Redwood Shores: Oracle.
- [40] Y. Chen, M. Lawford, H. Wang and A. Wassylng, "Insulin Pump Software Certification," in *Foundations of Health Information Engineering and System*, Macau, 2013.