2014-06-26

The Impact of Fuzzy Requirements on Medical Device Software Development

Martin McHugh
Technological University Dublin, martin.mchugh@tudublin.ie

Abder-Rahman Ali
Université d'Auvergne

Fergal McCaffery
Dundalk Institute of Technology, Dundalk, Ireland.

Recommended Citation

McHugh, M., Ali, A.R & McCaffery, F. (2014). The Impact of Fuzzy Requirements on Medical Device Software Development, *European Systems and Software Process Improvement and Innovation Conference EuroSPI* Luxembourg,25.-27, June. doi:10.21427/r0ws-x837

Funder: Science Foundation Ireland

The Impact of Fuzzy Requirements on Medical Device Software Development

Martin McHugh[1], Abder-Rahman Ali[2] and Fergal McCaffery[1]

Regulated Software Research Centre, Department of Computing and Mathematics
Dundalk Institute of Technology, Co. Louth Ireland
[1] *(Martin.McHugh, Fergal.McCaffery)@dkit.ie*
[2] *abder-rahman.a.ali@ieee.org*

Abstract

Any software development project can experience difficulties with unclear or vague requirements. Unfortunately, this problem can be experience two fold in regulated environments such as the medical device software development industry. In the medical device software development industry, development organisations must contend with vague or “fuzzy” both the customer and regulatory bodies. As new requirements are introduced they can have a knock on effect on other requirements. These requirements should be analysed to determine if they are conflicting, cooperative, mutually exclusive and irrelevant. Only when the requirement is classified can a clear method be established as how to integrate that requirement with previous ones. Medical device software organisations could benefit from understanding the impact of fuzzy requirements as it could result in reduced rework at a later stage in the project.

Keywords

Requirements Engineering, Medical, Fuzzy Requirements, FDA

1 Introduction

Every software development project consist of a Requirements phase. It is at this phase it is established what is to be development. Experience suggests that requirements are the biggest software engineering problem for the developers of large, complex systems. Many decades after the invention of computer programming, software practitioners still have raging debates about exactly what a “requirement” actually consists [1]. A software requirement can be defined as: *“a software capability needed by the user to solve a problem or to achieve an objective”; “a software capability that must be met or possessed by a system or a system component to satisfy a contract, standard, specification, or other formally imposed documentation*” [2].

It is generally agreed that the goal of the requirements phase is to establish what the software must do without describing how to do it. Most authors agree in principle that requirements should specify “what” rather than “how”. In other words, the goal of requirements is to understand and specify the problem to be solved rather than the solution. The most basic reason for this is that a specification in terms of the problem, captures the actual requirements and does not over constrain the subsequent design or implementation. Also, solutions are typically more complex, more difficult to change, and harder to understand than a specification of the problem [3].

Obtaining good software requirements is a crucial step towards building reliable and usable software systems. Studies show that one of the main reasons for software project failures is due to poor requirements [4]. It is extremely desirable to detect errors in the requirements before the design and development of the software begins. Due to the nature of the requirements specification phase, there is a lot of room for misunderstanding and committing errors, and it is quite possible that the requirements specification does not accurately represent the client's needs [5].

2 Medical Device Software Development

Medical device software is typically developed in accordance with the V-Model [6]. When developing software in accordance with the V-Model each stage of development is completed sequentially. Unlike other plan driven software development life cycles such as the Waterfall model, testing is planned in conjunction with each stage of development. The V-Model is typically followed as it produces the necessary deliverables required when seeking regulatory approval. However, there is a shift towards more agile development techniques in the medical device software development industry [7-9]. Agile methods appear to solve an often faced problem when following a plan driven SDLC, i.e. accommodating changing requirements once the requirements phase has been completed. However, this flexibility can create problems in itself. When following a plan driven approach the requirements are heavily refined before development begins, this includes resolving issues where requirements are unclear. When following agile methods, requirements are subject to change at any point in a software development project, therefore the process of understanding fuzzy requirements is need throughout a software development project. Medical device software development organisations who wish to market their device for use must conform to the regulations within that region. For example, medical devices marketed for use must beer the CE mark, showing conformance, and those marketed for use within the United States (US) must provide evidence of conformance to the Food and Drug Administration (FDA)

3 FDA & IEC 62304 stance on Requirements

The FDA regulations impose stringent requirements on the process by which software systems used in medical devices are developed. These requirements translate into various software artefacts that must be made available for the software to be FDA compliant [10] and, for medical device software, the FDA is responsible for assuring that the device utilizing the software is safe and effective [1].

FDA requires medical device manufacturers to submit their device requirements before beginning development. System and software requirements are taken from the FDA medical device Quality System Regulation [11]. FDA regulations cover all aspects of the medical device product lifecycle, and the FDA requires medical device manufacturers to submit evidence of product safety and efficacy for FDA review and clearance before the manufacturer can market, sell, or distribute the product [1]. Thus, it is critical to obtain information from the FDA on the requirements applicable to the proposed device [5].

Validation compares the final product to the original specifications [3], and is closely related to the requirements specification. You can validate the user's requirements; this is where ambiguity reigns most of the time and where formal methods, through the use of specification languages, have the biggest strides. There is still a wide gap between what the user wants and what the developer understands that the user wants. Very often this is where one of the causes of initial system failures can be found [12]. Software validation is the confirmation that all software requirements have been met and that all software requirements are traceable to the system requirements, provided that it is not possible to validate software without predetermined and documented software requirements [13]. There are two major types of validation that come into play with medical devices - design validation and process validation. Design validation means establishing, by objective evidence, that device specifications conform to the user's needs and the device's intended uses. Process validation, on the other hand, means establishing, by objective evidence, that a process consistently produces the desired result or a product meeting the predetermined specifications [14]. The FDA requires medical device manufacturers to submit their device specifications before beginning development. Thus, validation could come at early stages of development if the user's requirements could be precisely defined, and which from them the rest of the development derived [15]. Ideally, validation work would be accomplished while the requirements are being written [12]. Any safety and regulatory requirements for medical devices necessarily call for rigorous software development methods to ensure reliability and to protect public health. In addition to that, requirements and specifications based on medical practice are needed to help ensure that devices will perform appropriately [16].

The regulatory bodies request that medical device software development organizations clearly demonstrate how they follow a software development life cycle without mandating a particular life cycle. In order to comply with the regulatory requirements of the medical device industry, it is necessary to have clear linkages to traceability from requirements through the different stages of the software development and maintenance life cycles. Traceability is central to medical device software development and essential for regulatory approval. Software traceability refers to the ability to describe and follow the life of a requirement in both forward and backward direction [17]. FDA for instance states that traceability analysis must be used to verify that a software design implements all of its specified requirements [18]. Thus, traceability is particularly important for medical device companies, as they have to demonstrate this in order to achieve FDA compliance [19].

IEC 62304:2006 [20] is harmonized with the European Medical Device Directive (MDD) [16] and is approved for use by the FDA. As with guidance documents, adherence to IEC 62304:2006 is not mandatory, however, if a manufacturer chooses not to follow it, they would need to provide a sufficient explanation behind not following it. IEC 62304:2006 does not address software development lifecycle models; instead, it defines processes, which consist of activities that should be conducted in each medical device software development project [21]. As with the QSR, initial reading of IEC 62304:2006 would appear to suggest it should be followed in accordance with a sequential lifecycle model such as Waterfall Model. The publishers of IEC 62304:2006 observed that the standard appeared to mandate following the Waterfall Model and added the following to remove any ambiguity;

"it is easiest to describe the processes in this standard in a sequence, implying a "waterfall" or "once through" life cycle model. However, other life cycles can also be used"

4 Fuzzy Requirements

Requirements are sometimes not specified and documented in detail in many software development projects, which makes software validation and maintenance very difficult. One challenge is that many product requirements are fuzzy in nature. Actually, customers usually describe their requirements in fuzzy terms such as *good*, *high*, *very important*, *etc.* Translating such fuzzy terms into design specifications that will accurately create the desired product is difficult [12].

There are two important goals in requirements engineering: (a) acquiring requirements that are satisfactory to their customers; and (b) generating feasible requirements. These two goals often compete with each other. To achieve both goals, the requirements often need to be refined many times [12].

4.1 Fuzzy Requirements & Fuzzy Sets

In the medical device software domain, fuzzy requirements may emerge. An example of such requirements is:

R: the software system should fully support the clinician

The constraint imposed by the fuzzy requirement R can be represented as a satisfaction (membership) function, denoted as Sat_R, which maps an element of $R's$ domain D to a number in the range $[0,1]$, which represents how well the requirement is satisfied [7]:

$$Sat_R : D \rightarrow [0,1] \qquad (1)$$

Let us assume that the type of medical device software used is a medical imaging system. The elasticity of R

can be captured using the satisfaction function, and corresponds to the membership function of the fuzzy set *FULLY* in the requirement R [7].

Examples of the characteristics which should be available in order for the software system to be considered as a support for the clinician are as follows:

C_1: load medical image
C_2: view medical image
C_3: segment medical image
C_4: save medical image
…
C_n

A membership function of the fuzzy set *FULLY* that can be used, is Zadeh's *S-function*, defined as follows:

$$\mu_F(x) = \begin{cases} 0, x \le a; \\ 2\left(\frac{x-a}{c-a}\right)^2, a \le x \le b; \\ 1-2\left(\frac{x-c}{c-a}\right)^2, b \le x \le c; \\ 1, x \ge c \end{cases}$$

(2)

Where $\mu_F(x)$ is the degree of membership of the requirement x (represented in terms of the numbers of characteristics achieved, provided that the weights of importance of the characteristics is assumed to be the same) in the fuzzy set *FULLY*, where the value evaluates to the range $[0,1]$, such that, if no characteristics are available, $\mu_F(x) = 0$. a is the *minimum* number of characteristics, c is the maximum number of characteristics, and b is any value between a and c. The S-function can be plotted as shown in figure.1.

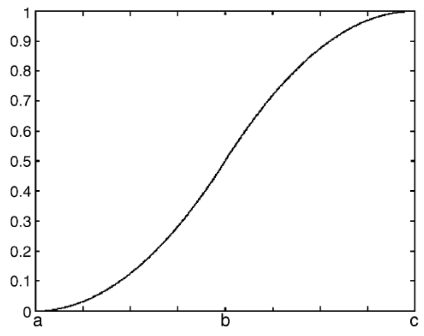

Figure 1. Membership Function of the Fuzzy Set F (support)

5 Relationship Classification

There are four types of significant relationships between requirements: (a) conflicting; (b) cooperative; (c) mutually exclusive; and (d) irrelevant. The classification is determined by how satisfying one requirement impacts the satisfaction degree of another requirement [12].

Two requirements are *conflicting* if raising satisfaction in one requirement *often* decreases the other's level of satisfaction. If it *always* decreases the satisfaction degree of the other, they are said to be *completely conflicting*. Figure 2 shows an example of completely and partially conflicting requirements [22].

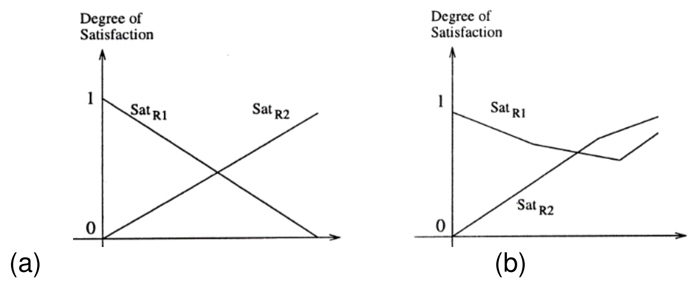

Figure 2. (a) completely conflicting requirements; (b) Partially conflicting requirements

Fuzzy conflicting relationships can relax the conditions of the crisp conflicting relationships using fuzzy terms such as *strong*, *medium*, *weak*, *etc*. Thus, one can define terms such as *strong conflict*, *medium conflict*, and *weak conflict* using satisfaction functions. Figure 3 shows an example of fuzzy conflicting relationships [22], where it can be noticed that when two requirements have the conflicting degree 0.5, we are very sure that they are weak conflicting, since their satisfaction degree in the membership function *Weak Conflict* is 1.0, and are not strong conflicting since their degree of satisfaction in membership function *Strong Conflict* is 0. These two requirements are somewhat medium conflicting since their degree of satisfaction in membership function *Medium Conflict* is 0.6.

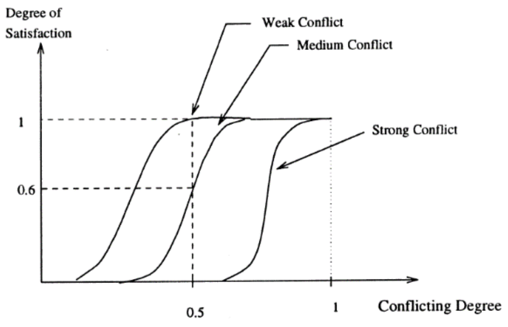

Figure 3 Fuzzy conflicting requirements

Two requirements are *cooperative* if increasing the satisfaction in one often increases the degree in the other. If the rise in satisfaction of one always increases satisfaction in the other, they are *completely cooperative*. Figure 4 shows an example of completely and partially cooperative requirements [22]. Fuzzy cooperative relationships can relax the conditions of the crisp conflicting relationships using fuzzy terms such as *strong*, *medium*, *weak*, *etc*. Thus, one can define terms such as *strong cooperative*, *medium cooperative*, and *weak cooperative* using satisfaction functions.

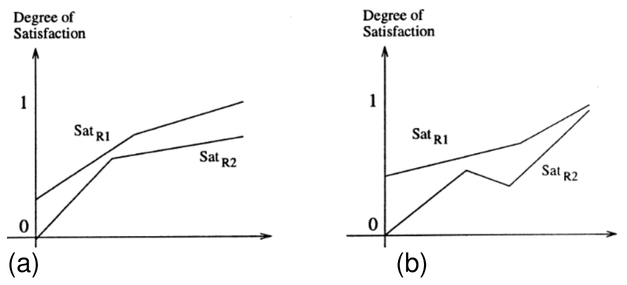

Figure 4 (a) completely cooperative requirements; (b) Partially cooperative requirements

Sometimes, two requirements cannot be satisfied at the same time, such that, if one fuzzy requirement is satisfied, the other is not satisfied at all. Those requirements are referred to as *mutually exclusive requirements [12]*.

6 Implicit Relationships Detection

In large scale software systems for instance, many conflicts are implicit, and thus, difficult to identify. Therefore, it helps to have techniques that can aid in identifying implicit conflicting and cooperative relationships between requirements. In this case, several heuristics can be used to infer relationships between requirements based on the identified relationships [22, 23]:

Heuristic rule 1 (infer relationships from cooperative requirements): *Let D be a domain shared between three requirements R1, R2, and R3. If requirement R1 is completely cooperative with R2 in D, R2 is completely cooperative with R3 in D, and they are not irrelevant, then R1 is completely cooperative with R3 in D.*

Heuristic rule 2 (infer relationships from conflicting and cooperative requirements): *Let D be a domain shared between three requirements R1, R2, and R3. If requirement R1 is completely cooperative with R2 in D, R2 completely conflicts with R3 in D, and they are not irrelevant, then R1 is completely conflicting with R3 in D.*

Heuristic rule 3 (infer relationships from conflicting requirements): *Let D be a domain shared between three requirements R1, R2, and R3. If requirement R1 completely conflicts with R2 in D, R2 completely conflicts with R3 in D, and they are not irrelevant, then R1 is completely cooperative with R3 in D.*

It can be noticed from heuristic rule 1 that a completely cooperative relationship in a domain is transitive. Whilst heuristic rule 3 indicates that a completely conflicting relationship in a domain is *not* transitive.

7 Results and Discussions

Suppose that we are planning to develop some medical device software, MEDSYS. Such software to be used in or as a medical device is subject to user requirements. However, unlike unregulated software, medical device software must meet both the user's requirements and the requirements of the regulatory body (*i.e.* FDA) of the region into which the software will be marketed [10]. Thus, we are expected to comply with both user requirements and regulatory requirements.

Examples of user requirements for MEDSYS are:

R_1: The medical device software shall *fully* support the clinician
R_2: The medical device software shall be developed in *short* time

Examples of IEC 62304:2006 requirements for MEDSYS are:

R_3: The manufacturer shall retain *sufficient* records to permit the test to be repeated
R_4: The manufacturer shall establish procedures to ensure that the released software product can be *reliably* delivered to the point of use without corruption or unauthorized change
R_5: The manufacturer shall consider potential causes including, as appropriate, *reasonably* foreseeable misuse

In the above requirements, the fuzzy terms have been written in italics. Such fuzzy terms can be characterized by fuzzy sets, and thus, represented by a membership function.

Figure 5 shows the relationships between the requirements as given by a requirements analyst and a customer, where "-" denotes a conflictive relationship, and "+" denotes a cooperative relationship. Here, we assume that the conflictive and cooperative relationships are *complete* (Figure 2(a) and Figure 4(a)).

	R_1	R_2	R_3	R_4	R_5
R_1			+	+	+
R_2				-	
R_3					
R_4					
R_5					

Figure 5 Initial relationships as specificed by a requirements analyst and a customer

Using the heuristics in section 6, more relationships (shown with a green background box) could be infered as shown in figure 6.

	R_1	R_2	R_3	R_4	R_5
R_1		🟩 -	+	+	+
R_2				-	
R_3		🟩 -		🟩 +	🟩 +
R_4			🟩 +		🟩 +
R_5		🟩 -	🟩 +	🟩 +	

Figure 6 Inferring relationships between requirements

From Figure 6, we can notice some interesting relationships being inferred. Since most of the time we may be interested in trying to manage between the user requirements and the requirements of the IEC 62304:2006 standard (regulatory requirements), it is thus necessary to find out where such requirements would not meet (*i.e.* conflict). For instance, it can be noticed that the user requirement R_2 and the IEC 62304:2006 requirement R_5 cannot be achieved at the same time, since they completely conflict with each other.

8 Conclusions

Vague or unclear software requirements also known as “Fuzzy Requirements” can have a detrimental effect on a software development project. Often what is finally delivered to the customer is not what they asked for, rather what the software development organization perceived them to need. This problem can be exacerbated in the medical device software development industry where there are two customers, the end user and the regulatory bodies. Regulatory bodies impose strict controls to ensure the safe and reliable performance of medical devices. However, these regulations and associated development standards introduce requirements which can be deemed as fuzzy. By fully understanding fuzzy and categorizing them they can be accommodated better in a software development project and therefore the potential for a project being deemed a failure can be reduced.

Acknowledgments

This research is supported by the Science Foundation Ireland (SFI) Stokes Lectureship Programme, grant number 07/SK/I1299, the SFI Principal Investigator Programme, grant number 08/IN.1/I2030 (the funding of this project was awarded by Science Foundation Ireland under a co-funding initiative by the Irish Government and European Regional Development Fund), and supported in part by Lero - the Irish Software Engineering Research Centre (*http://www.lero.ie*) grant 10/CE/I1855.

9 References

[1] F. McCaffery, V. Casey, M. Sivakumar, G. Coleman, P. Donnelly, and J. Burton, "Medical Device Software Traceability," in *Software and Systems Traceability*, J. Cleland-Huang, O. Gotel, and A. Zisman, Eds., ed: Springer-Verlag, 2012.

[2] C. Denger, R. L. Feldman, M. Host, C. Lindholm, and F. Schull, "A Snapshot of the State of Practice in Software Development for Medical Devices," presented at the First International Symposium on Empirical Software Engineering and Measurement, 2007. ESEM 2007, Madrid, 2007.

[3] R. C. Fries, *Reliable Design of Medical Devices* vol. 3rd ed. Boca Raton FL: CRC, 2012.

[4] H. Mehrfard, H. Pirzadeh, and A. Hamou-Lhadj, "Investigating the Capability of Agile Processes to Support Life-Science Regulations: The Case of XP and FDA Regulations with a Focus on Human Factor Requirements," presented at the In Proceedings of SERA (selected papers), 2010.

[5] F. Pavese and A. B. Forbes, *Data Modeling for Metrology and Testing in Measurement Science*: Birkhäuser, 2009.

[6] T. H. Faris, *Safe And Sound Software: Creating an Efficient And Effective Quality System for Software Medical Device Organizations*: Asq Press, 2006.

[7] J. K. Shapiro, "The Pathway to Market for your Medical Device: A Primer on Obtaining Information from FDA " *FDLI,* vol. Update May/June, 2008.

[8] M. S. Sivakumar, V. Casey, F. McCaffery, and G. Colema, "Verification & Validation in Medi SPICE," presented at the The 11th International SPICE Conference Process Improvement and Capability dEtermination, Dublin, 2011.

[9] J. O. Grady, *System Requirements Analysis* Amsterdam: Elsevier Academic, 2006.

[10] D. Farb and B. Gordon, *Pharmaceutical Computer Validation Introduction Guidebook*: UniversityOfHealthCare, 2005.

[11] C. T. DeMarco, *Medical Device Design and Regulation*: Asq Press, 2011.

[12] A. Dasso and A. Funes, *Verification, Validation and Testing in Software Engineering*: Idea Group Pub., 2007.

[13] V. Casey and F. McCaffery, "Med-Trace: Traceability Assessment Method for Medical Device Software Development.," presented at the European Systems & Software Process Improvement and Innovation Conference, (EuroSPI). Roskilde, Denmark, 2011.

[14] J. Cleland-Huang, A. Czauderna, M. Gibiec, and J. Emenecker, "A machine learning approach for tracing regulatory codes to product specific requirements," presented at the Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, Cape Town, South Africa, 2010.

[15] F. McCaffery and G. Coleman, "The need for a software process improvement model for the Medical Device Industry," *International Review on Computers and Software,* vol. 2, pp. 10-15, 2007.

[16] FDA, "Title 21--Food and Drugs Chapter I --Food and Drug Administration Department of Health and Human Services subchapter h--Medical Devices part 820 Quality System Regulation," ed: *U.S. Department of Health and Human Services*, 2007.

[17] *FDA Design Control Guidance for Medical Device Manufacturers*, 1997.

[18] FDA, "General Principles of Software Validation: Final Guidance for Industry and FDA Staff," ed: *Centre for Devices and Radiological Health*, 2002.

[19] AAMI, "ANSI/AAMI/IEC 62304, Medical device Software - Software life cycle processes," ed. Association for the Advancement of Medical Instrumentation, 2006.

[20] *Directive 2007/47/EC of the European Parliament and of the Council of 5 September 2007*, 2007.

[21] M. McHugh, F. McCaffery, and V. Casey, "Standalone Software as an Active Medical Device " presented at the The 11th International SPICE Conference Process Improvement and Capability dEtermination, Dublin, 2011.

[22] C. Lan and B. Ramesh, "Agile Requirements Engineering Practices: An Empirical Study," *Software, IEEE,* vol. 25, pp. 60-67, 2008.

[23] M. Coram and S. Bohner, "The impact of agile methods on software project management," in *Engineering of Computer-Based Systems, 2005. ECBS '05. 12th IEEE International Conference and Workshops on the*, 2005, pp. 363-370.

[24] Standish Group, "Chaos Report," ed, 1995.

[25] F. Paetsch, A. Eberlein, and F. Maurer, "Requirements engineering and agile software development," presented at the Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on, 2003.

[26] P. L. Jones, J. Jorgens, A. R. T. Jr, and M. Weber, "Risk Management in the Design of Medical Device Software Systems," *Biomedical Instrumentation & Technology: July 2002,* vol. 36, pp. 237-266, 2002.
[27] (2001). *Manifesto for Agile Software*. Available: http://agilemanifesto.org/
[28] M. McHugh, F. McCaffery, and V. Casey, "Barriers to using Agile Software Development Practices within the Medical Device Industry," in *European Systems and Software Process Improvement and Innovation Conference, EuroSPI* Vienna Austria, 2012.
[29] M. Cohn, *Succeeding with Agile - Software Development Using Scrum*. Upper Saddle River NJ: Addison Wesley, 2011.
[30] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic Mapping Studies in Software Engineering," presented at the 12th International Conference on Evaluation and Assessment in Software Engineering (EASE), University of Bari, Italy, 2008.

10 Author CVs

Martin Mc Hugh

Martin is a lecturer in Computer Science at Dublin Institute of Technology. He received his B.Sc. (Hons.) in Information Technology Management in 2005 and M.Sc. in Computer Science in 2009, from Dundalk Institute of Technology. He is now undertaking research for his Ph.D. in the area of software process improvement for medical devices with emphasis on the usage of agile practices when developing medical device software, as part of the Regulated Software Research Centre in Dundalk Institute of Technology.

Abder-Rahman Ali

Abder-Rahman received his BSc in Computer Science in 2006 from the University of Jordan and MSc Software Engineering in 2009 from DePaul University. He is interested in applying technology to medicine, and in building computer aided diagnosis systems that aid in diagnosing disease. He is currently pursuing his Ph.D. degree at Université d'Auvergne in France, as part of the ISIT lab, in the area of fuzzy clustering and discrete geometry for image analysis of MRI and ultrasound imaging sequences for Hepatocellular Carcinoma (HCC). He is very passionate to the idea of applying computer science to medical imaging, and software engineering to medical device software systems, in an eventual goal to come up with algorithms and systems that aid in Computer Aided Diagnosis (CAD). He also likes to adopt fuzzy logic in his research.

Fergal Mc Caffery

Dr Fergal Mc Caffery is the leader of the Regulated Software Research Centre in Dundalk Institute of Technology and a member of Lero. He has been awarded Science Foundation Ireland funding through the Stokes Lectureship, Principal Investigator and CSET funding Programmes to research the area of software process improvement for the medical device domain. Additionally, he has received EU FP7 and Enterprise Ireland Commercialisation research funding to improve the effectiveness of embedded software development environments for the medical device industry.