Software Process Improvement Roadmaps – Using Design Patterns to Aid SME's Developing Medical Device Software in the Implementation of IEC 62304

Peter Rust(✉), Derek Flood, and Fergal McCaffery

Regulated Software Research Centre and Lero,
Dundalk Institute of Technology, Dundalk, Ireland
{peter.rust,derek.flood,fergal.mccaffery}@dkit.ie

Abstract. One stated objective of the European Union is to encourage SME's expand their area of operation into other domains. The medical device domain is one such domain identified by the EU. Medical device software development must be carried out in a manner that compliance with certain medical device standards and regulations can be demonstrated. *IEC 62304, Medical device software - software life cycle processes*, is a standard that defines the processes that are required to be executed in order to develop safe software. SME software development organizations wishing to expand their operations into the medical device software development domain face serious challenges in demonstrating compliance with IEC 62304. The standard describes the set of processes, activities, and tasks that are required to be carried out, but importantly do not describe how they should be carried out. This paper describes the development of a roadmap that will aid software development SME's, entering the medical device software development domain, by the use of design patterns to generate "How-to" artefacts, overcome the challenge of demonstrating compliance.

Keywords: SME's · Medical device software · Medical device standards · Regulatory compliance · Software roadmap · Software process improvement · Software process improvement roadmaps · IEC 62304 · Design patterns

1 Introduction

In Europe, the medtech industry generates over €100 billion annually and employs approximately 575,000 people. As many as 95 % of these companies are small to medium enterprises (SME) [1]. SME software development organizations wishing to enter the medical device software development arena must be able to demonstrate that the processes, utilised in the development of the software, are compliant with IEC 62304 [2]. IEC 62304 is the Medical Device Software – software life cycle processes standard and it is harmonised in the European Union (EU) and the United States of America (USA). The path to regulatory compliance for software development organizations that are able to demonstrate compliance with IEC 62304 is shorter and they will be able to market their product both in the EU and USA. However, implementing

P.M. Clarke et al. (Eds.): SPICE 2016, CCIS 609, pp. 43–56, 2016.
DOI: 10.1007/978-3-319-38980-6_4

IEC 62304 within a medical device software development organization is not straightforward or easy. Höss et al. [3] describe a pilot project that they undertook to acquire skills in implementing IEC 62304 in a hospital-based environment (in-house manufacture). They concluded that the pilot project carried out at their facility clearly demonstrated that the interpretation and implementation of IEC 62304 is not feasible without appropriately qualified staff. They recognized that it could be carried out by a small team with limited resources although the initial effort is significant and a learning curve must be overcome.

Standalone software can now be classified as an active medical device [4]. IEC 62304 defines the life cycle requirements for medical device software. These requirements vary according to the safety classification assigned to the software system as defined by requirement 4.3. The software safety class is assigned according to the possible effects on the patient, operator, or other people resulting from a hazard to which the software system can contribute. The software safety classes are based on severity as follows:

Class A: no injury or damage to health is possible

Class B: non-serious injury is possible

Class C: death or serious injury is possible.

The life cycle requirements establish a common framework for medical device software life cycle processes, but critically the standard does not state how the processes should be implemented. SME's in the general software development sphere will have their own processes in place; however these may not be robust enough to satisfy the requirements of IEC 62304. The processes in place will require improvement and new processes will inevitably be required to be implemented.

The Capability Maturity Model® Integration (CMMI®) [5] and ISO/IEC 15504-5:2012 (SPICE) [6] are two software process improvement models that are directed at the general software development domain. These models are not robust enough to allow organizations achieve medical device regulatory compliance [7]. The development of MDevSPICE® (formerly known as Medi SPICE) has filled this void [7]. An MDevSPICE® assessment will identify the gaps that appear in an organization's processes, but critically, not how to fill these gaps.

During the initial research period, various methods were considered as a means for bringing the IEC 62304 standard to SME's developing medical device software. These included decision trees, flowcharts, roadmaps and design patterns. After a review of the literature and due consideration of the various methods, a roadmap was chosen as the most appropriate method. A roadmap was subsequently developed for the implementation of IEC 62304. During the process of developing the "how-to" artefacts that form a crucial part of the roadmap, it was noted that design patterns may have a role in developing these artefacts. This paper explores that avenue.

The remainder of this paper is structured in the following manner: Sect. 2 describes roadmaps and the roadmapping process. Section 3 introduces the concept of design patterns, outlining their history, the process of creating them and describing their structure. Section 4 presents a discussion while Sect. 5 presents the conclusions and future work.

2 Roadmaps and the Roadmapping Process

2.1 Development of the Roadmap

The definition of a roadmap for standards implementation has gone through a number of versions before the following was decided upon: "*A series of Activities, comprised of Tasks that will guide an organisation, through the use of specific "How-to's" towards compliance with regulatory standards*". To generate the roadmap for IEC 62304 the roadmap development method described by Flood et al. [8] has been applied. This method, described below, has been revised in light of the latest definition of a roadmap.

1. Identify requirements of the standard and rephrase them as Tasks;
2. Group the Tasks into logical Activities;
3. Order the Activities into a sequence by which they can be introduced into an organization in a rational manner;
4. Validate the generated roadmap;
5. Identify the "How-To's" that can meet the identified Tasks;
6. Validate the "How-To's" in a host organization.

Flood et al. [8, 9] have already applied the roadmapping process to ISO 14971 and IEC 62366 and these roadmaps have been validated with industry experts. A roadmap has also been developed for traceability in the medical device domain and now with the development of an IEC 62304 roadmap, the suite is nearing completion.

2.2 Process Used to Develop the Roadmap

In step 1 as described above the standard was decomposed into its elementary requirements and a total of 172 elementary requirements were identified. The requirements were then transformed into Tasks by the application of an action verb.

Taking as an example of the transformation process, IEC 62304, requirement 5.1.1 states that "*the manufacturer shall establish a software development plan (or plans) for conducting the activities of the software development process appropriate to the scope, magnitude, and software safety classifications of the software system to be developed*." This was transformed into a Task defined as **"***establish a software development plan. The plan is for the software system to be developed (for conducting the eight activities of the software development process), defining fully the software lifecycle model to be utilised.***"**

In step 2 when the transformation of all the requirements was complete, the Tasks were analysed for particular keywords that would aid their grouping into logical Activities. The above Task was assigned the keyword "Planning". A total of seven Tasks were initially grouped according to this keyword and an Activity created titled "Software Development Planning". The generation of the final roadmap is described in Rust et al. [10]. Following the roadmap generation process, a total of five Tasks were allocated to the Software Development Planning Activity. The final allocation of Tasks to Activities is detailed in Table 1.

Table 1. Final number of tasks per activity

Ref	Activity	Final No of Tasks
1	QMS	1
2	RMS	1
3	Software Safety Classification	3
4	Software Development Planning Process	5
5	Software Configuration Management Process	4
6	Software Risk Management Process	9
7	Software Requirements Analysis Process	4
8	Software Architectural Design Process	10
9	Software Detailed Design Process	4
10	Software Unit Implementation and Verification Process	5
11	Software Integration and Integration Testing Process	6
12	Software System Testing Process	3
13	Software Release Process	6
14	Software Problem Resolution Process	8
15	Change Request Process	7
16	Software Maintenance Process	6

2.3 Structure of the Roadmap

The metaphor for the roadmap is detailed in Fig. 1 below. The metaphor presented below was designed to highlight the stage at which each of the Activities may be applied during the development of a medical device software project. It can be seen that a number of the processes may be ongoing for the duration of the software development process.

Each of the phases in the software development lifecycle is depicted to overlap as a number of Tasks may be performed in parallel. Taking an example of the Software Unit Implementation and Verification Process and the Software Detailed Design Process, it is feasible that during the second Task of the Software Detailed Design Process – "*Document a design with enough detail to allow correct implementation of each software unit*", the organization may commence the first Task of the Software Unit Implementation – "*Implement each software unit*".

Each task will be associated with an artefact in the "How-to" repository and will also be context dependent (Safety Classification). The artefacts will be constructed using design patterns as a building mechanism.

3 Design Patterns

3.1 What is a Design Pattern?

Christopher Alexander is an architect that introduced the concept of design patterns for the design of towns, communities, buildings and homes [11]. He compiled a catalogue of some 253 patterns to describe various elements and combinations of elements for

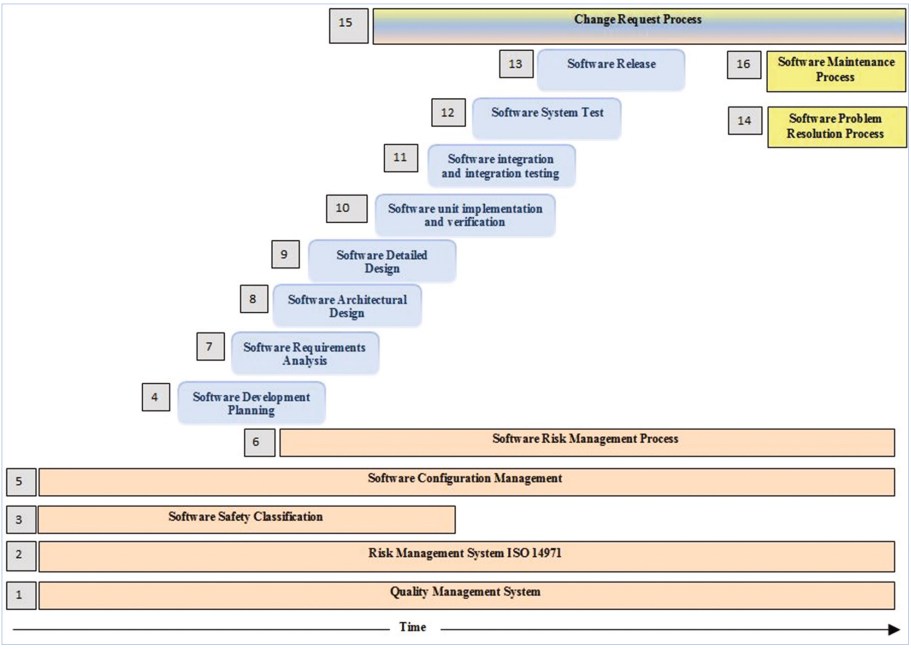

Fig. 1. Metaphor for the Roadmap

repeatable architectural design. Alexander stated, "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice" [12].

While Alexander was talking about patterns for the design and construction of towns, houses and buildings, the concept can be applied to the design and "construction" of the "How-to" artefacts for standards implementation. Indeed, design patterns have been proposed for the following other domains:

- Object orientated software design [12].
- Business Process Modelling [13].
- Real time and embedded systems [14].
- Security risk orientated patterns [15].
- Architectural design patterns (computer) [16].
- Workflow patterns [17].
- Nurse practitioner practice patterns [18].
- Behaviour design patterns [19].
- Patterns for effective use cases [20].
- Pro HTML5 and CSS3 design patterns [21].

3.2 History of Design Patterns

Christopher Alexander describes in his "*book of two halves*", A Pattern Language [11] and A Timeless Way of Building [22], how first the gate must be constructed and on passing through the gate the practice of building can be undertaken. Building the gate comprises the identification of the design patterns that constitute the language. Once that is complete the patterns are used to design and build concrete structures that are unique to the user of the language. The patterns are based on real life experience and can be reused time and time again without replication of a particular solution.

OOPSLA – Object-Oriented Programming, Systems, Languages, and Applications held its first conference in 1986 [23] and about 50 papers were presented. Among the presenters were Ralph Johnson, Ward Cunningham and Kent Beck. In 1987 Ward Cunningham and Kent Beck were working as consultants on a Smalltalk project that was having difficulty in completing [24]. Both had an interest in Alexander's design patterns. Alexander believed that the occupiers of a building should be involved in its design. Cunningham and Beck decided to allow the users of the software to complete the design. Cunningham developed a five pattern "language" that helped these novice designers take advantage of Smalltalk's strengths and avoid its weaknesses. The experiment was successful in that the project was completed.

Erich Gamma while conducting doctoral research became intrigued by Alexander's work and the reusability feature of design patterns. In August of 1993 [25], Kent Beck and Grady Booch sponsored a hillside retreat for a group of people who were also interested in pattern languages and wanted to build on Gamma's work. Present were Ward Cunningham, Ralph Johnson, Ken Auer, Hal Hilderbrand, Grady Booch, Kent Beck and Jim Coplien. The Hillside Group was established. In 1994 [24] they sponsored a conference on the Pattern Language of Programs (PloP-94) which was held at Allerton Park in Monticello, Illinois, a property of the University of Illinois at Urbana Champaign. The conference chair was Ralph Johnson and in the program chair, Ward Cunningham. The PloP conferences are held annually to this day. The use of design patterns in software engineering has greatly increased over the years. The book Design Patterns: Elements of Reusable Object-Oriented Software has sold over 500,000 copies since it was first published.

3.3 Process Used to Develop Design Patterns

A literature review conducted by the authors has unearthed various methods of developing Pattern Languages. This section describes a number of the more prominent methods and provides a justification of the methods selected to develop the Language.

Alexander et al. [11] describes the format adopted in writing their patterns as:

- First, there is a picture, which shows an archetypal example of the problem.
- Second, after the picture, each pattern has an introductory paragraph which sets the context for the pattern, by explaining how it helps to complete certain larger patterns.

- The essence of the problem is then described in one or two sentences.
- The body of the problem, which is the longest section, describes the empirical background of the pattern, the evidence for its validity and the range of ways it can be manifested in the building.
- The solution follows in the form of an instruction – so that you know exactly what you need to do, to build the pattern.
- The solution in the form of a labelled diagram follows indicating its main components.
- Lastly, the links to other patterns in the language are listed.

Gamma et al. [12] start by describing the four essential elements of a pattern:

1. **Name:** The pattern name is a handle we can use to describe a design problem, its solutions, and consequences in a word or two.
2. **Problem:** The problem describes when to apply the pattern.
3. **Solution:** The solution describes the elements that make up the design, their relationships, responsibilities, and collaborations.
4. **Consequences:** The consequences are the results and trade-offs of applying the pattern.

Each pattern is then described using a consistent format:

- **Pattern Name and Classification** - The pattern's name conveys the essence of the pattern. The pattern's classification falls under two headings purpose and scope.
- **Intent** - What does the design pattern do? What is its rationale and intent? What particular design issue or problem does it address?
- **Also Known As** - Other well-known names for the pattern, if any.
- **Motivation** - A scenario that illustrates a design problem and how the class and object structures in the pattern solve the problem.
- **Applicability** - What are the situations in which the design pattern can be applied?
- **Structure** - A graphical representation of the classes in the pattern using a notation based on the Object Modelling Technique (OMT).
- **Participants** - The classes and/or objects participating in the design pattern and their responsibilities.
- **Collaborations** - How the participants collaborate to carry out their responsibilities.
- **Consequences** - How does the pattern support its objectives? What are the trade-offs and results of using the pattern?
- **Implementation** - What pitfalls, hints, or techniques should you be aware of when implementing the pattern? Are there language-specific issues?
- **Sample Code** - Code fragments that illustrate how you might implement the pattern in C++ or Smalltalk.
- **Known Uses** - Examples of the pattern found in real systems.
- **Related Patterns** - What design patterns are closely related to this one?

Wellhausen and Fießer [26] offer their advice in writing patterns using the door lock as an example. Although they do not reference Alexander or Gamma, the pattern produced is very similar to that of the patterns proposed by Alexander and Gamma,

starting first with the name and then with a picture or diagram followed by the context, problem, forces, solution and consequences. The sequence in which the design pattern should be addressed is detailed in Fig. 2 below.

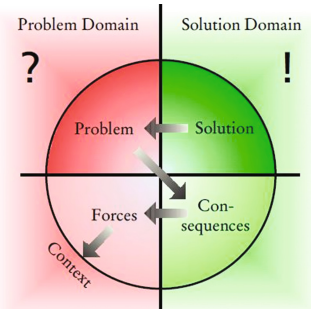

Fig. 2. Essential pattern sections and their writing order

Cunningham [27] provides the following tips for writing a pattern language:

- Pick a whole area.
- Make a list of all the things that you have learned about the area.
- Cast each item on your list as a solution.
- Write each item as a pattern. Include four paragraphs in total, the first two separated from the last two by the word "therefore".
- Organise them into sections, writing a short introduction to the section and listing each pattern in the section.
- Write an introduction to the language that hints at the forces that you will be addressing.

Arising out of a workshop at PLoP-95, Meszaros and Doble [28] collaborated on writing a pattern language for pattern writing. These patterns were reviewed at Plop-96 and were published in 1997.

There are recurring themes running throughout all of these contributions. With this in mind and that the patterns to be created are for standards implementation, the following format is being adopted:

- Name
- Software Safety Classification
- Context
- Problem
- Motivation (Forces)
- Solution
- Consequences
- Related Patterns
- ISO/IEC Reference
- Catalogue (Section)
- Alias

3.4 Structure of the Design Patterns

To demonstrate the process of writing a pattern language, using the tips for writing pattern languages proposed by Cunningham [27], and referencing the pattern development process described by Wellhausen and Fießer [26], the following steps were undertaken:

1. The area chosen is standards implementation in the medical device software development domain.
2. The standard to be implemented is IEC 62304 [6].
3. The requirements of IEC 62304 have been listed and cast as solutions.
4. The solution to the problem is to establish a software development plan.
5. The problem for which a pattern is to be written is "The software developer wants to demonstrate compliance with requirement 5.1.1 - Establish a Software Development Plan - for software classified as Class A."

Solution: The solution will describe the main essential elements that are required. The solution is not particular or specific because a pattern is like a template that can be applied in many different situations. In this instance, the solution is the actual software development plan for medical device software with a safety classification of Class A. In previous research, a software development plan template was designed to take account of all safety classifications A, B and C. This content of this template has been stripped down to include only those elements that are essential for compliance with IEC 62304 under a Class A safety classification. The design pattern solution for medical device software with a safety classification of Class B will only contain the additional elements that are required to achieve compliance for this safety classification. Similarly, the solution for medical device software with a safety classification of Class C will only contain those remaining elements that are required to complete a full software development plan for Class C software.

Problem: The problem describes when to apply the pattern. It explains the problem and its context. The problem for the software development team is that compliance with IEC 62304 must be demonstrated. A physical software development plan must be produced. IEC 62304 lists certain elements that must be addressed or referenced by the plan. For SME's with little or no experience in the medical device domain this can be a daunting task. The design pattern will not only define the problem but critically will provide a template for the solution.

Consequences: The consequences will describe the benefits and possible liabilities of using the pattern. Consider first the benefits of using the pattern, a plan for the medical device software development is established. The plan is safety classification dependent. Only those requirements that are necessary are considered. The software development team has identified only those activities of the software development process that are required for the particular safety classification and by means of the roadmap will be able to address these in a timely manner.

Secondly the liabilities must be considered. Planning is an iterative process. Failure to keep the plan up-dated may have detrimental consequences. The software development

plan will include or reference certain other plans that are required by other activities and processes. As the software development proceeds, these plans will evolve and quite possibly change. Keeping the software development plan updated and circulated to all members of the team will ensure that every member of the team knows what artefacts are current. Failure may lead to members of the team using outdated artefacts.

Motivation (Forces): The reasons why the solution is required. The motivation will mirror the consequences. A software development plan is a necessary artefact to demonstrate compliance with IEC 62304. The software development plan requires that certain identified tasks be undertaken. The resources required to undertake the development must be identified and put in place. Failure to plan and failure to resource the development adequately will result in failure to achieve regulatory compliance.

Context: What is the situation when the pattern can be applied? List the elements that have already been completed. Referencing the roadmap, it can be established which elements that will be already underway at the point in time that the software development plan is required to be in place. The quality management, risk management and software configuration management processes will have commenced. The software system will have been assigned a safety classification of A, B or C. The level of planning required will vary depending on the safety classification assigned. In this instance a safety classification of Class A has been assigned, so only the design pattern for Class A software development plan need be considered.

Name: The pattern name is used to describe the problem, its solution and consequences in a short sentence. Naming a pattern in this way allows the user to easily find the correct pattern. The context can be established and work can start on the solution. The name chosen in this instance is - Establish Software Development Plan for a safety classification of Class A.

Catalogue: The catalogue will be used to group patterns in a meaningful way, so that the user can quickly see other patterns that may be useful. In this case the design pattern will be filed under the Planning entry in the catalogue.

Related Patterns: 5.2. Update the software development plan. 5.4. Identify supporting items. 5.5. Plan configuration items management control. X.X Documentation Plan.

IEC References: IEC 62304 requirement 5.1.1 The manufacturer shall establish a software development plan (or plans) for conducting the activities of the software development process appropriate to the scope, magnitude, and software safety classifications of the software system to be developed. The software development life cycle model shall either be fully defined or be referenced in the plan (or plans).

Alias: There are no known aliases at this time.

Artefact: The template for the software development plan for medical device software with a safety classification of Class A, which is compliant with IEC 62304, comprises twenty pages. Only those elements that are required by specific requirements are included. The elements are gathered in sections as detailed Rust et al. [29]. The contents page is reproduced hereunder (Fig. 3).

The Final Pattern: The final pattern comprises all the eleven sections described above, and detailed in Fig. 4 below.

Contents

Fig. 3. Contents page from software development plan template Class A

Element	Structure
Name	Establish a Software Development Plan
Software Safety Classification	Class A
Context	The software system has been assigned a safety classification of A, B or C. The level of planning required varies depending on the safety classification. In this instance the safety classification is Class A.
Problem	The Manufacturer and hence the software developer wants to demonstrate compliance with requirement Clause 5.1.1 for software classified as Class A.
Motivation/Forces	A software development plan is a necessary artefact to demonstrate compliance with IEC 62304. The software development plan requires that certain identified tasks be undertaken. The resources required to undertake the development must be identified and put in place.
Solution	Establish a software development plan that complies with IEC 62304 for the designated safety classification assigned to the software system. In this instance the safety classification is Class A.
Consequences - Benefits & Liabilities	Benefits: A plan for the software development is established. The software developer has identified the activities of the software development process and will be able to address these in a timely manner. Liabilities: Planning is an iterative process. Failure to update the plan could have detrimental consequences.
Artefact - (The How To) Template	Software Development Plan Template Ref 5.1 Class A
Links to Related Patterns	5.2. Update the software development plan. 5.4. Identify supporting items. 5.5. Plan configuration items management control. X.X Documentation Plan
IEC Reference	5.1.1 The manufacturer shall establish a software development plan (or plans) for conducting the activities of the software development process appropriate to the scope, magnitude, and software safety classifications of the software system to be developed. The software development life cycle model shall either be fully defined or be referenced in the plan (or plans).
Alias	None known

Fig. 4. The final pattern

4 Discussion

IEC 62304 defines the processes required for the development of safe software for the medical device domain but does not tell the organization "how to" implement the processes. The generated roadmap, together with the "how-to" artefacts, when completed will fill this gap. The "how-to" artefacts will be developed using design patterns as described above. Each step on the roadmap will reference an appropriate design pattern that will contain the basic information that will guide the software development team in the implementation of the process. The software development team need only choose those processes which they require, which in turn will only reference the design patterns that are relevant to the software safety class of the medical device software. The software development team will be familiar with the concept of design patterns and will therefore be better positioned to understand and use the design patterns presented to them. The combination of the roadmap and the design patterns will guide the software development team in the production of the artefacts that will aid them in demonstrating compliance with the regulatory requirements.

5 Conclusions and Future Work

One of the stated objectives of the EU is to encourage software development SME's to enter other domains. Medical device software development is one such domain. However, the medical device software development domain is strictly regulated. The regulatory standards provide a description of all the necessary processes that must be planned for, executed and that the results of the execution are recorded and documented. The documents will be audited and compliance with the regulations and standards will be determined by the results of the audit. How to plan, execute and document their processes is the challenge for SME's entering the medical device software domain. The design patterns that identify the "how-to" element of the processes combined with the roadmap, will guide the SME's along the path to regulatory compliance in a timely and planned manner.

The roadmap is currently being validated by industry experts. The next stage of this work is to create more design patterns to help build additional "How-to" artefacts and have them trialled in SME medical device software development organizations that are new to the medical device domain.

Acknowledgement. This research is supported by the Science Foundation Ireland Principal Investigator Programme, grant number 08/IN.1/I2030 and by Lero - the Irish Software Research Centre (http://www.lero.ie) grant 10/CE/I1855 & 13/RC/20194.

References

1. About the Medtech sector | IMDA. http://www.imda.ie/Sectors/IMDA/IMDA.nsf/vPages/Medtech_sector~about-the-medtech-sector!OpenDocument. Accessed 23 Feb 2016

2. IEC: IEC 62304:2006 - Medical device software – Software life cycle processes. ISO, Geneva, Switzerland (2006)
3. Höss, A., Lampe, C., Panse, R., Ackermann, B., Naumann, J., Jäkel, O.: First experiences with the implementation of the European standard EN 62304 on medical device software for the quality assurance of a radiotherapy unit. Radiat. Oncol. **9**, 10 (2014)
4. McHugh, M., McCaffery, F., Casey, V.: Standalone software as an active medical device. In: O'Connor, R.V., Rout, T., McCaffery, F., Dorling, A. (eds.) SPICE 2011. CCIS, vol. 155, pp. 97–107. Springer, Heidelberg (2011)
5. CMMI Product Team (2010), Capability Maturity Model® Integration for Development Version 1.2. Software Engineering Institute, Pittsburch, PA (2010)
6. ISO/IEC: ISO/IEC 15504-5, Information technology - Process Assessment - Part 5: An Exemplar Process Assessment Model. ISO, Geneva, Switzerland (2012)
7. McCaffery, F., Dorling, A.: Medi SPICE development. Softw. Process Maint. Evol. Improv. Pract. J. **22**(4), 255–268 (2010)
8. Flood, D., McCaffery, F., Casey, V., McKeever, R., Rust, P.: A roadmap to ISO 14971 implementation. J. Softw. Process Evol. **27**(5), 319–336 (2015)
9. Flood, D., McCaffery, F., Casey, V., Regan, G.: A methodology for software process improvement roadmaps for regulated domains – example with IEC 62366. In: McCaffery, F., O'Connor, R.V., Messnarz, R. (eds.) EuroSPI 2013. CCIS, vol. 364, pp. 25–35. Springer, Heidelberg (2013)
10. Rust, P., Flood, D., McCaffery, F.: Software process improvement & roadmapping – a roadmap for implementing IEC 62304 in organizations developing and maintaining medical device software. In: Rout, T., O'Connor, R.V., Dorling, A. (eds.) SPICE 2015. CCIS, vol. 526, pp. 19–30. Springer, Heidelberg (2015)
11. Alexander, C., Ishikawa, S., Silverstein, M.: A Pattern Language: Towns, Buildings, Construction. OUP USA (1977)
12. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Pearson Education, Upper Saddle River (1994)
13. Atwood, D.: BPM process patterns: repeatable design for BPM process models. BP Trends, May 2006
14. Douglass, B.P.: Doing Hard Time: Developing Real-time Systems with UML, Objects, Frameworks, and Patterns. Addison-Wesley Professional, Boston (1999)
15. Ahmed, N., Matulevičius, R.: Securing business processes using security risk-oriented patterns. Comput. Stand. Interfaces **36**(4), 723–733 (2014)
16. Schmidt, D.C., Stal, M., Rohnert, H., Buschmann, F.: Pattern-Oriented Software Architecture. Patterns for Concurrent and Networked Objects. Wiley, New York (2013)
17. van Der Aalst, W.: Workflow patterns. Distrib. Parallel Databases **14**, 5–51 (2003)
18. Fain, J., Melkus, G.: Nurse practitioner practice patterns based on standards of medical care for patients with diabetes. Diabetes Care **17**, 879–881 (1994)
19. Taylor, G., Wray, R.: Behavior design patterns: engineering human behavior models. Ann Arbor (2004)
20. Adolph, S., Cockburn, A., Bramble, P.: Patterns for effective use cases (2002)
21. Bowers, M., Synodinos, D., Sumner, V., Zack, A.: Pro HTML5 and CSS3 Design Patterns (2011)
22. Alexander, C.: The Timeless Way of Building - Christopher Alexander. OUP USA (1979)
23. OOPSLA. http://www.oopsla.org/oopsla-history/. Accessed 25 Feb 2016
24. History of Patterns. http://www.c2.com/cgi/wiki?HistoryOfPatterns. Accessed 04 Feb 2016
25. The Hillside Group - Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/The_Hillside_Group. Accessed 26 Feb 2016

26. Wellhausen, T., Fießer, A.: How to write a pattern. In: European Conference on Pattern (2011)
27. Cunningham, W.: Tips For Writing Pattern Languages (1994). http://www.c2.com/cgi/wiki?TipsForWritingPatternLanguages. Accessed 02 Feb 2016
28. Meszaros, G., Doble, J.: A pattern language for pattern writing. Pattern Lang. Progr. Des. **3**, 529–574 (1998)
29. Rust, P., Flood, D., McCaffery, F.: Creation of an IEC 62304 compliant Software Development Plan. In: EuroasiaSPI (2015)