RESEARCH PROJECT REPORT

EMSOFT

September 2019

Research Project: **Investigate standards driven IoT product lines**

Investigator: Alexandre Jouen, French engineering intern
Auckland University of Technology

Prepared by: Alexandre Jouen

Statement of Purpose

This study aims to examine what software design method should be used for IoT product lines in order to follow as close as possible quality requirements.

An efficient software design allows rapid handling and orderly handling of the various product functions and ensures the security of user information. In addition, code maintainability will allow developers to save time during the software development phase as well as for software updates. Finally, an efficient software design prevents cyber-attacks which is a huge problem in our current society.

Research Questions

The study examines three major research questions:

What current industry standards apply to software design for IoT product lines?

What are the current software design formalisms best suited for IoT product lines?

What software design formalism(s) could be used to monitor the requirements of the standards as closely as possible?

Study Methodology

The objective of the research was to determine the best way to design an IoT product lines software according to current software design formalisms and standards requirements.

The methodology of the study consisted first in researching on both sides, the software design standards adapted to IoT product lines and the design methods commonly used for the design of these software. The objective was then to link the two parties to find the design method that best meet the requirements of the standards.

The findings can be used by AUT university like specifications during the IoT product lines software designing phase.

Report format

1 Current industry standards which apply to the design of IoT product lines software.

As a first step, a key words search has been done in order to preselect several standards.

ISO/IEC 15288-2015	Systems and software engineering -- System life cycle processes
IEEE STD 1074-2006	Standard for Developing a Software Project Life Cycle Process
IEEE STD 1016-2009	Standard for Information Technology--Systems Design--Software Design Descriptions
IEEE STD 1734-2011	Standard for Quality of Electronic and Software Intellectual Property Used in System and System on Chip (SoC) Designs
IEEE STD 1872-2015	Standard Ontologies for Robotics and Automation
IEEE STD 3006.8-2018	Recommended Practice for Analysing Reliability Data for Equipment Used in Industrial and Commercial Power Systems
IEEE STD C37.1-2007	Standard for SCADA and Automation Systems
IEEE STD C37.115-2003	Standard Test Method for Use in the Evaluation of Message Communications Between Intelligent Electronic Devices in an Integrated Substation Protection, Control and Data Acquisition System
IEEE STD C37.240-2014	IEEE Standard Cybersecurity Requirements for Substation Automation, Protection, and Control Systems
ISO/IEC/IEEE 42010	Systems and software engineering -- Architecture description
ISO/IEC/IEEE 42020	Software, systems and enterprise -- Architecture processes
IEEE Std 1379-2000(R2006)	IEEE Recommended Practice for Data Communications Between Remote Terminal Units and Intelligent Electronic Devices in a Substation
ISO/IEC 26552:2019	Software and systems engineering -- Tools and methods for product line architecture design

Table 1: Standards preselection (Key words: SPL, Software, IoT, Security, Industrial CPS, Industrial automation systems, Automation)

Then, the next step was to read all standards abstracts and check many reviews dealing with the subject in order to eliminate non-usable standards.

According to the review *"On the Evolution of Software and Systems Product Line Standards"* [4]:

The reference model for Software and System Product Line (SSPL) defines software and systems product line engineering and management with two life cycles (domain engineering and application engineering) and two process groups (organizational management and technical management). Figure 1 provides the reference model for all interrelated standards in SSPL. The reference model also provides interrelationships between the domain and application engineering life cycles and provides guidelines to adapt these lifecycles to a variety of organizational and technical environments for different quality and business goals. Organizational and technical management process groups focus on helping organizations to establish and improve capabilities for nurturing their product lines from conception to retirement and for establishing and managing relationships with customers, providers, and other key stakeholders. The SSPL standards can be used:

• by organizations intended to implement product lines – to understand, adopt, and enact the processes, tools, and methods for [product line requirements engineering, architecture design, realization, verification and validation, technical management, organizational management]

• by tool vendors to develop tools with required capabilities for supporting the entire life cycle of software product line engineering

SSPL is a complex endeavour requiring a diversified range of tools and methods for industrial adoption. To address these needs, an entire set of proposed standards (ISO/IEC 26550 to 26599) are envisioned with few standards published, few under development and few yet to be developed. On the other hand, SSPL is still an emerging area witnessing a significant progress in terms of new tools and methods. This requires extensive study of evolving market requirements and will lead to continuous evolution of the standards as well. We list some of the core standards as follows:

• The product line reference model (ISO/IEC 26550) provides an overview of the consecutive International Standards (i.e., ISO/IEC 26551 through ISO/IEC 26556) as well as the structure of the model.

• Processes and capabilities of methods and tools for product line scoping, domain requirements engineering, and application requirements engineering are described by *ISO/IEC 26551, Software and systems engineering – Tools and methods for product line requirements engineering.*

The rest of the standards follow the similar pattern to describe processes and capabilities of methods and tools for the entire software product line engineering life cycle.

• *ISO/IEC 26552, Software and systems engineering – Tools and methods for product line architecture design*.

• *ISO/IEC 26553, Software and systems engineering – Tools and methods for product line realization*.

• *ISO/IEC 26554, Software and systems engineering – Tools and methods for product line verification and validation*.

• *ISO/IEC 26555, Software and systems engineering – Tools and methods for product line technical management*.

• ISO/IEC *26556, Software and systems engineering – Tools and methods for product line organizational management*.

In addition, there are several draft standards currently in progress for variability mechanisms (ISO/IEC 26557), variability modelling (ISO/IEC 26558) and variability traceability (ISO/IEC 26559).

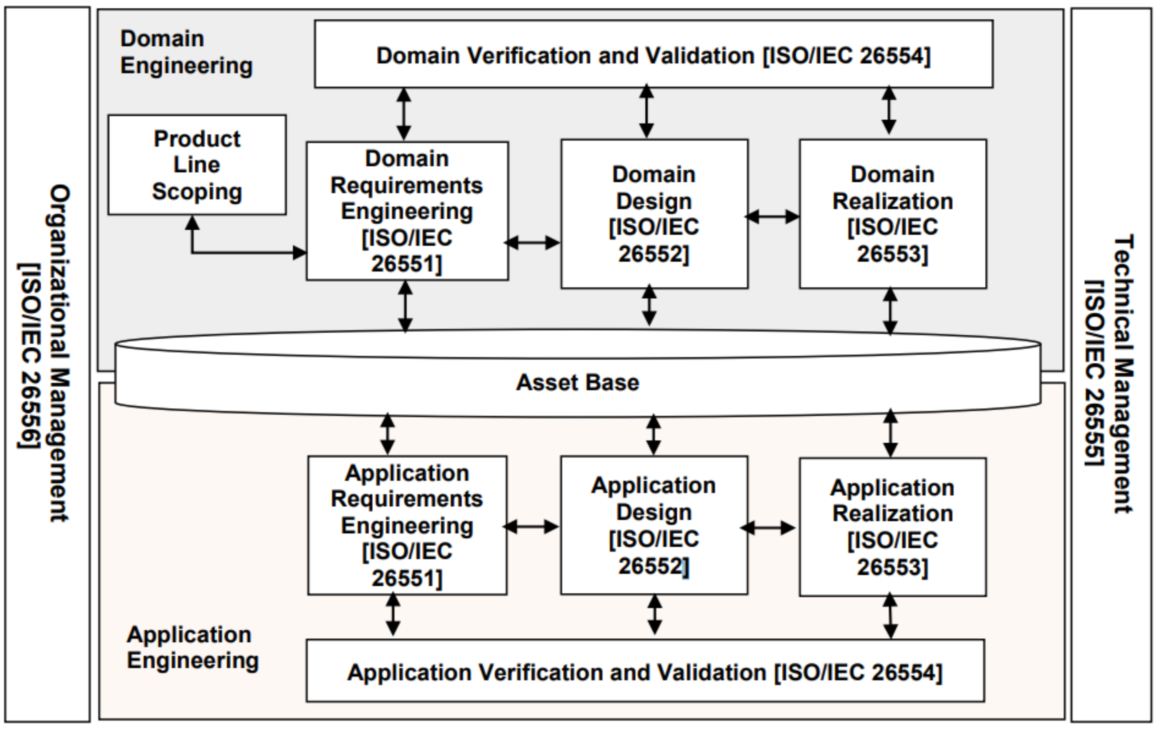

Domain Engineering
Domain Verification and Validation [ISO/IEC 26554]
Product Line Scoping
Domain Requirements Engineering [ISO/IEC 26551]
Domain Design [ISO/IEC 26552]
Domain Realization [ISO/IEC 26553]
Asset Base
Application Requirements Engineering [ISO/IEC 26551]
Application Design [ISO/IEC 26552]
Application Realization [ISO/IEC 26553]
Application Engineering
Application Verification and Validation [ISO/IEC 26554]
Organizational Management [ISO/IEC 26556]
Technical Management [ISO/IEC 26555]

Figure 1: Reference model for software and systems product line adapted from

In this panel of standards, we will focus only on the design part: *ISO/IEC 26552, Software and systems engineering – Tools and methods for product line architecture design [5].*

This standard, within the context of methods and tools for architecture design for software and systems product lines:

— defines processes and their subprocesses performed during domain and application architecture design. Those processes are described in terms of purpose, inputs, tasks and outcomes;

— defines method capabilities to support the defined tasks of each process;

— defines tool capabilities to automate/semi-automate tasks or defined method capabilities.

According to ISO/IEC/IEEE, the common standard usable for software engineering is *ISO/IEC/IEEE 15288-2015 Systems and software engineering-System life cycle processes* [8].

This Internal Standard applies to the full life cycle of systems, including conception, development, production, utilization, support and retirement of systems, and to the acquisition and supply of systems, whether performed internally or externally to an organisation.

Finally, it seems that the *ISO/IEC 26552:2019* [5] standard is an adaptation of several standards including *ISO/IEC/IEEE 15288:2015* [8] but applicable to product lines software design.

2 Current software design formalisms best suited for IoT product lines.

According to the review "*Comparison of Software Product Line Architecture Design Methods: COPA, FAST, FORM, KobrA and QADA"* [3]:

Product line architectures (PLAs) have been under continuous attention in the software research community during the past few years. Although several methods have been established to create PLAs there are not available studies comparing PLA methods. Five methods are known to answer the needs of software product lines: COPA, FAST, FORM, KobrA and QADA.

The first of the methods mentioned, a Component- Oriented Platform Architecting Method for product family engineering, i.e. COPA, is a component-oriented but architecture-centric method that enables the development of software intensive product families.

FAST – Family-Oriented Abstraction, Specification and Translation - is a software development process focused on building families.

Feature-Oriented Reuse Method for product line software engineering, FORM is an extension to the FODA method. The core of FORM lies in the analysis of domain features and the use of these features to develop reusable and adaptable domain artifacts. That is, FORM is a feature-oriented approach to product line architecture engineering.

Kobra is an acronym for Komponentenbasierte Anwendungsentwicklung, denoting a practical method for component-based product line engineering with UML.

Quality-driven Architecture Design and Analysis, shortly QADA states a product line architecture design method providing traceable product quality and design time quality assessment.

However, they don't compete with each other, because each of them has a special goal or ideology. All the methods highlight and follow this ideology throughout the method descriptions.

- **COPA**. Concentrated on balancing between topdown and bottom-up approaches and covering all the aspects of product line engineering i.e. architecture, process, business and organization.
- **FAST.** Family oriented process description with activities, artifacts and roles. Therefore, it is very adapting but not applicable as it is.
- **FORM.** Feature-oriented method for capturing commonality inside a domain. Extended also to cover architectural design and development of code assets.
- **KobrA**. Practical, simple method for traditional component-based software engineering with UML. Adapts to both single systems and family development.
- **QADA.** Concentrated on architectural design according to quality requirements. Provides support for parallel quality assessment of product line software architectures.

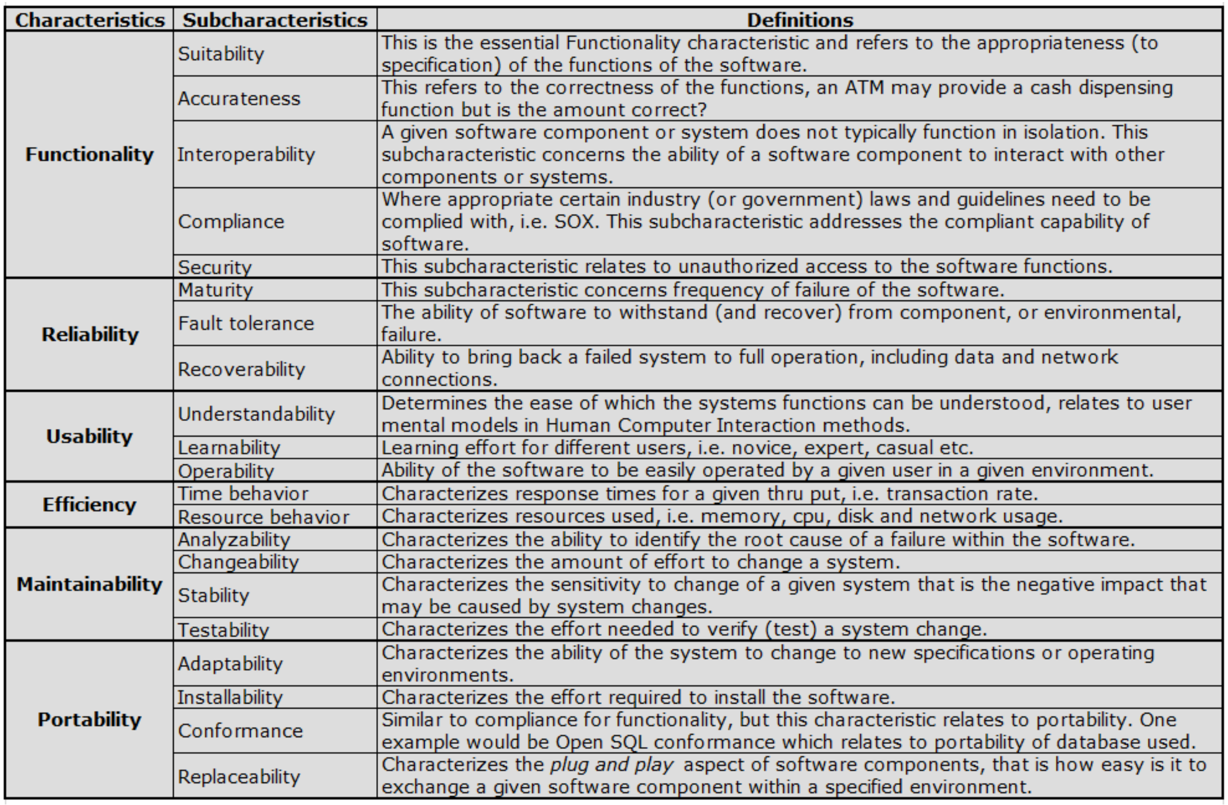
3 Analysis of one industry standard which apply the best to IoT product lines

3.1 *ISO/IEC 26552:2019* Classification of requirements based on qualities

We have already seen that the requirements of the standard *ISO/IEC 26552* [5] are extracted from other standards such as *ISO/IEC/IEEE 15288* [8] or *ISO/IEC/IEEE 42020* [7].

The main advantage is that all requirements are already adapted to product lines field.

According to *ISO/IEC 9126 Software engineering — Product quality* [10], the table below highlights all qualities an efficient software must have.



Characteristics	Subcharacteristics	Definitions
Functionality	Suitability	This is the essential Functionality characteristic and refers to the appropriateness (to specification) of the functions of the software.
	Accurateness	This refers to the correctness of the functions, an ATM may provide a cash dispensing function but is the amount correct?
	Interoperability	A given software component or system does not typically function in isolation. This subcharacteristic concerns the ability of a software component to interact with other components or systems.
	Compliance	Where appropriate certain industry (or government) laws and guidelines need to be complied with, i.e. SOX. This subcharacteristic addresses the compliant capability of software.
	Security	This subcharacteristic relates to unauthorized access to the software functions.
Reliability	Maturity	This subcharacteristic concerns frequency of failure of the software.
	Fault tolerance	The ability of software to withstand (and recover) from component, or environmental, failure.
	Recoverability	Ability to bring back a failed system to full operation, including data and network connections.
Usability	Understandability	Determines the ease of which the systems functions can be understood, relates to user mental models in Human Computer Interaction methods.
	Learnability	Learning effort for different users, i.e. novice, expert, casual etc.
	Operability	Ability of the software to be easily operated by a given user in a given environment.
Efficiency	Time behavior	Characterizes response times for a given thru put, i.e. transaction rate.
	Resource behavior	Characterizes resources used, i.e. memory, cpu, disk and network usage.
Maintainability	Analyzability	Characterizes the ability to identify the root cause of a failure within the software.
	Changeability	Characterizes the amount of effort to change a system.
	Stability	Characterizes the sensitivity to change of a given system that is the negative impact that may be caused by system changes.
	Testability	Characterizes the effort needed to verify (test) a system change.
Portability	Adaptability	Characterizes the ability of the system to change to new specifications or operating environments.
	Installability	Characterizes the effort required to install the software.
	Conformance	Similar to compliance for functionality, but this characteristic relates to portability. One example would be Open SQL conformance which relates to portability of database used.
	Replaceability	Characterizes the *plug and play* aspect of software components, that is how easy is it to exchange a given software component within a specified environment.

Table 2: The full table of Characteristics and Sub characteristics for the ISO 9126:1991 Quality Model

Below, a classification of *ISO/IEC 26552* [5] requirements based on the above qualities.

Portability	Develop models and views of the domain architecture
	Develop models of the application specific architecture
Maintainability-Reliability	Assess the application specific architecture documentation
	Assess the domain architecture documentation for structure and texture
	Trace the usage status of variability mechanism category in architecture
	Identify architectural artefacts managed as domain assets
	Deploy capabilities and resources for architecture enablement
	Analyse problem space of the domain architecture
	Analyse common rules guiding realization

Maintainability-Reliability	Analyse domain architecture and assess stakeholder satisfaction
	Refine external variability into internal variability
	Trace the usage status of variability mechanism category in architecture
	Analyse application architecture and assess stakeholder satisfaction
	Close and prepare for the architecture management plan change
	Maintain variability model in architecture
Functionality	Analyse problem space of the domain architecture
	Domain architecture evaluation
	Cross functional validation and verification
	Classifying firm and evolving set of architecturally significant functional requirements
	Access the functional and quality requirements specification
	A set of architecturally significant functional requirements
	Identify architectural artefacts managed as application assets
	Monitor and assess compliance with governance directives and guidance
Usability	Integrating lessons learned during architecture management
	Store lessons learned into the permanent storage for the further use
	Access technical requirements, technical readiness level and lessons learned in the single system development or previous SSPL
Efficiency	Trace the usage status of variability mechanism category in architecture
	Enabling technology support
	Confirming that variability mechanisms in architecture (e.g. variability mechanisms in codes) can correctly address the defined variability dependencies
	Confirming that variability mechanisms in architecture can correctly address the defined constraints
	Providing evaluation algorithm for verifying whether a variability mechanism confirms the defined variability dependencies and constraints
	Verifying the testability of variability mechanisms used in architecture.
	Visualize variability mechanisms included in architecture
	Verify whether the variability mechanism properly supports the defined variability including dependencies and constraints;
	Refer the testability of variability mechanisms used in architecture

Table 3: Classification of requirements based on qualities (ISO/IEC 26552:2019)

4 Design method that can be used to link requirements from standards more closely with actual software designs

After studying the different methods for the software design of product lines, it seems that the method that best meets our study requirements is the QADA method.

Quality Driven Method: Quality-driven Architecture Design and quality Analysis (QADA) is a traceable quality-based method to design and evaluate software architecture. QADA contains scenario-based quality analysis to evaluate if the architecture design options meet the quality requirements. QADA consists of three viewpoints: structural view, behaviour view, and deployment view at two levels of abstractions: conceptual level and concrete level. Quality attributes are categorized to related views and each view has associated targets on the two levels at certain design phase.

As a quality-driven method, quality analysis is performed at both levels with different attentions. Analysis on conceptual level focuses on variability analysis and architectural analysis by quality-based methodology. Quality analysis of concrete architecture emphasizes on the customer value analysis and scenario-based quality analysis. The purpose of architectural analysis at conceptual level is to provide a knowledge base for a more comprehensive quality attributes analysis at the concrete level.

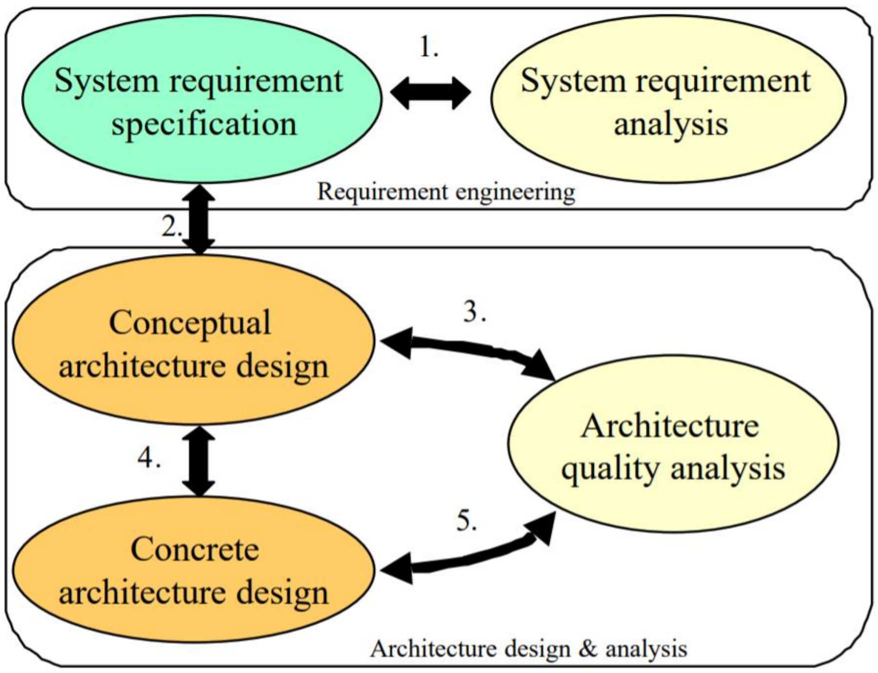

System requirement specification
1.
System requirement analysis
Requirement engineering
2.
Conceptual architecture design
3.
Architecture quality analysis
4.
5.
Concrete architecture design
Architecture design & analysis

Figure 2: QADA method main phases

All the information about QADA method can be found in the review *"Quality-driven architecture design and quality analysis method: A revolutionary initiation approach to a product line architecture"* [2].

5 Conclusion

In this research report, we have studied the different standards and design methods that apply to IoT product lines.

First, a keyword search allowed to pre-select standards.

Then the review reading on the subject led to the identification of the appropriate ISO/IEC 26552:2019 standard, which is based on other known standards such as ISO/IEC/IEEE 15288:2015.

In addition, a classification of the requirements of this standard based on quality has been carried out with the aim of saving future designers time.

After, the reading of reviews dealing with software design methods for IoT product lines allowed us to highlight five different design methods.

Finally, one of the software design methods for the IoT product lines was selected: QADA method and a review explaining its use was highlighted.

This method will allow the person or persons using it to comply with the associated standards.

REFERENCES

[1] L. Tan, Y. Lin, and H. Ye, "Quality-Oriented Software Product Line Architecture Design," *J. Softw. Eng. Appl.*, vol. 05, no. 07, pp. 472–476, 2012.

[2] M. Matinlassi, E. Niemelä, and L. Dobrica, "Quality-driven architecture design and quality analysis method: A revolutionary initiation approach to a product line architecture," *VTT Publ.*, no. 456, pp. 3–128, 2002.

[3] M. Matinlassi, "Comparison of Software Product Line Architecture Design Methods: Mari Matinlassi," *26th Int. Conf. Softw. Eng.*, 2004.

[4] S. Chimalakonda *et al.*, "On the Evolution of Software and Systems Product Line Standards," *ACM SIGSOFT Softw. Eng. Notes*, vol. 41, no. 3, pp. 27–30, 2018.

[5] I. Standard, "Software and systems engineering – Tools and methods for product line architecture design STANDARD ISO / IEC 26552/2019", 2019.

[6] I. Standard, "Systems and software engineering — Architecture description STANDARD ISO / IEC / IEEE 42010:2010", 2010.

[7] I. Standard, "Systems and software engineering — Architecture processes STANDARD ISO / IEC / IEEE 42020:2019", 2019.

[8] I. Standard, "Systems and software engineering — Systems life cycle processes STANDARD ISO / IEC / IEEE 15288:2015", 2015.

[9] I. Standard, "Systems and software engineering — Software life cycle processes STANDARD ISO / IEC / IEEE 12207:2008", 2008.

[10] I. Standard, "Software engineering — Product quality
STANDARD ISO / IEC 9126:1991", 1991.