Software Development Guidelines based on Fuzzy Decision-Making for Health care Applications

Elisabetta Ronchieri
INFN CNAF
elisabetta.ronchieri@cnaf.infn.it

Valerio Venturi
INFN CNAF
valerio.venturi@cnaf.infn.it

Cristina Vistoli
INFN CNAF
cristina.vistoli@cnaf.infn.it

Abstract

Software development of applications, regarding also health care, must deal with the needs of developers with different background. Incorporating the voice of developers, most often shared geographically, into the various phases of the software life cycle is becoming always and always important. Therefore, decision-making process is a valid way for translating software development needs into relevant guidelines to drive developers during their activity in order to obtain a maximum portability, extensibility and maintainability of software itself. The process takes into account multiple preferences and different tools in order to guarantee a certain freedom to developers making software development easy to achieve.

In this research, we define a fuzzy decision-making algorithm for software development guidelines. The algorithm contributes defining a set of guidelines, a set of goals, a cost function for guidelines, and a set of constraints. A fuzzy operator is used to aggregate goals and constraints. This algorithm was tested using realistic software development case studies applied to an health care application.

1. Introduction

Decision-making is a key in every part of a developer’s life to aim the definition, setting up and configuration of software. It consists of finding the best option from a feasible set. As a developer, you have to use your knowledge to make the best decisions you can make, weighting the positives and negatives of each option and considering all the alternatives. Some developers find that this process can be helpful in making their decisions and determining which option is the best of a given situation.

In literature there are several methods for making better decisions. One of which is based on the seven-step formula of the Josephson Institute of Ethics [22] applicable to developers as follows:

- **stop,** whatever is going on or you are doing, and **think,** about the future and how your decision will affect your software later;
- **clarify** software **goals** to know exactly where you are going with the decision you are trying to make;
- **determine results** considering what you know and figuring out what you need to know;
- **develop some options** selecting package versions, identifying build tools and version control systems;
- **consider the consequences** of your options and of the decision itself, identifying parts of your code that will be affected by the options or the decisions and how they will be affected;
- **make your decision**;
- **monitor** your decision and **modify** it if you are sure that what you made was wrong.

Another method [23] offers different techniques for decision-making such as prioritizing your decisions, evaluating which option is more important and deciding if a decision is worth making.

During software life cycle, it is crucial to guarantee the software quality and to reduce the development time and cost. Unfortunately software development deals with the uncertainty and vagueness from subjective perception and experience of developers in decision process. In the last decade, uncertainty is associated with decision-making research [18] that tries to sufficiently reduce uncertainty and doubt about alternatives to allow a reasonable choice to be made among them. Vagueness is mostly a problem of language, but it is not the main problem [31]. The objective of this research is to use a decision-making algorithm based on fuzzy set theory [32] to evaluate the selection process and to choose the best alternative from available alternatives. Fuzzy set theory incorporates human intelligence in decision-making models. It is used to represent the evaluation results [24], mainly because information is subjective, imprecise and expressed in a nature language by individual developers. By using fuzzy set theory,
decision problems can be represented mathematically as an optimization problem.

Fuzzy set theory provides a flexible framework, in which it is possible to satisfactorily solve many of the obstacles of lack of prevision. Fuzzy set theory represents the qualitative aspects in qualitative terms (i.e., linguistic terms) by means of linguistic variables, whose values are words or sentences in a natural or artificial language. In general, the use of words or sentences is a more direct, flexible and realistic form to express the qualitative aspects [8, 11, 12, 19]. However, there are situations, in which the framework is heterogeneous, i.e., the information is not equally important [10, 14, 20, 21]. For example, in the case of a group of developers expressing their opinion on the possible solution of a given problem, two aspects have to be modeled: 1) their decision must not be considered with equal relevance, given that, there will be developers with more experience than others, and therefore, all the opinions shall not be equally reliable; and 2) a final and global decision must be made using the initial and individual decision. The first aspect is modeled by assigning a weight to each developer. The weights may be interpreted in at least two different ways [14, 15]:

1. each developer is viewed as a sub-group and the weight reflects the relative size of this sub-group;
2. the weight may reflect the relevance of the developers in the group.

The second aspect is dealt with the usage of adequate operators for combining information before reaching a final decision, usually called information aggregation operators.

Health care systems, ranging from pharmaceutical research, medical imaging and drug development system, make use of applications to deliver health care to human community. Developers involved in the development of these applications need to face challenges related to compliance, cost effectiveness, and quality requirements to assure the application success in health care. Therefore, they have to use processes to conceive, design and develop health care applications, and to perform checks and control features to satisfy the specified requirements. Health care organizations have acknowledged the need for defining and implementing standards and processes [25]. For engineering the software, developers of health care applications, as developers of other applications in further domains, need to follow processes that are usually structured in several steps, for example related to the analysis of requirements, design, coding, testing, and configuration management [29], which strongly influence the success of the software project. Each step addresses specific needs and identifies a set of guidelines.

This paper proposes a generic algorithm that selects one guideline at time and only executes the described guidelines. This algorithm has been formulated having in mind two case studies identified from the experience of working with new infrastructures and software projects during their migration to ETICS (eInfrastructure for Testing, Integration and Configuration of Software) [4, 5]. The first case study aims to describe the usage and setting up of ETICS according to the special needs of an infrastructure. The second case study, instead, aims to collect guidelines related to the well-defined configuration and maintenance of software projects in order to facilitate the adoption of ETICS. However, some guidelines are always valid, not strictly related to the usage of ETICS itself, like the analyzing of the software structure, software dependencies, software versioning and software portability. In the latest three years, ETICS made experiences to estimate automatically software quality attributes like interoperability, portability, standardization and compliance: in particular, the interoperability verification between different standard implementation of DICOM (Digital Imaging and Communications in Medicine) was proved in [3, 13, 30]. Due to this activity, dicom4j (digital imaging and communications in medicine 4 java) is the health care application used to illustrate how the algorithm can be applied to this kind of applications.

This paper is organized as follows. In Section 2 an algorithm based on fuzzy decision-making adopted in this research is illustrated. Section 3 introduces a case study on a software development. Section 4 describes the health care application to which the algorithm has been applied, while Section 5 presents some results obtained by simulating the proposed algorithm. Finally Section 6 draws some conclusions.

2. Fuzzy decision-making algorithm

Fuzzy decision-making is proposed to deal with non-probabilistic uncertainty and vagueness in the environment in which the decision-making takes place. Goals of the decisions and imposed constraints represent two important elements of decision-making:

- **Goals of the decisions** are represented by the maximized objective function;
- **Imposed constraints** confine the decision space.

---

Fuzzy decision-making replaces the goals and the constraints with their fuzzy equivalents [32, 6].

In this section the usage of fuzzy decision-making to obtain an optimal decision in software development is described. The guideline that better satisfies the goals and constraints of the system is the optimal. The decision is often formulated as a quintuple $(A, \Theta, \Xi, k, D)$ [17], where:

- $A$ is the set of alternatives (i.e., guidelines);
- $\Theta$ is the set of the states of the environment in which the decision is taking place;
- $\Xi$ is the set of consequences that results from the choice of a guideline;
- $k$ maps $Ax\Theta \rightarrow \Xi$, specifying a resulting consequence for each element of the set of environment states $\Theta$ and each element of the set of alternatives $A$;
- $D$ is the decision function defined as $D: \Xi \rightarrow R$, reflecting the preference structure and incorporating the goals of the decision maker.

$D$ induces a preference ordering on the set of consequences $\Xi$ such that

$$\varepsilon_i \succ \varepsilon_j \text{ if and only if } D(\varepsilon_i) > D(\varepsilon_j)$$

where $\varepsilon_i, \varepsilon_j \in \Xi$ and $\succ$ is the preference relation expressing that consequence $\varepsilon_i$ is preferred to consequence $\varepsilon_j$.

The following definitions are fundamental to describe fuzzy decision-making algorithm. For a better understanding of the algorithm, $G$, which stands for guidelines, takes the place of $A$ in the quintuple.

### 2.1. Cost function

Let $G$ be a given set of possible guidelines, which contains a solution to a decision-making problem. Let $\Theta$ be the set of the states of the environment in which the decision is taking place.

A cost function $f$ can be defined as follows, associating for each guideline-states pair a consequence with $g \in G$ and $o_1, \ldots, o_m \in \Theta$:

$$f(g, o_1, \ldots, o_m) = \sum_{i,j=1}^{n,m} r_i(o_j)$$

where $r_i(o_j)$ is the cost of rule $i$ for guideline $g$, and states $o_1, \ldots, o_m$ represents all states relevant for the guideline $g$. The cost of rule $i$ can be expressed as

$$r_i = w_i d_i(o_j)$$

where $w_i$ is a constant factor, and $d_i(o_j)$ is an activation level for rule $i$ operating in fuzzy state $o_j \in \Theta$ with $j=1, \ldots, m$.

When all rules of guideline $g$ have an activation level tending to zero (i.e., the guideline $g$ is inactive), also the cost for this guideline will tend to zero:

$$\text{if } \sum_{i=1}^{n} d_i \rightarrow 0, f(g, o_1, \ldots, o_m) \rightarrow 0$$

In the case of an inactive guideline, a default rule described as follows needs to be added:

$$r_{n+1} = w_{n+1} d_{n+1}(o_j)$$

where $w_{n+1}$ is the value to which the cost function must tend when the activation level of the other rules tend to zero. The value $d_{n+1}$ can be computed as follows:

$$d_{n+1}(o_j) = 1 - \sum_{j=1}^{n,m} d_j(o_j)$$

Each fuzzy state is represented by a linguistic term. The activation level of a rule $i$, i.e., $d_i(o_j)$, can be considered as a predicate defined by a fuzzy aggregation operator over linguistic terms [16]. Predicates can be defined over a tree, considering root tree as a fuzzy aggregation operator, and tree leafs as fuzzy states using linguistic term.

### 2.2. Fuzzy goals

Let $G$ be a given set of possible guidelines which contains a solution to a decision-making problem. Let $\Theta$ be the set of the states of the environment in which the decision is taking place.

A fuzzy goal $GO$ is a fuzzy set on $f$ characterized by the membership function $\mu_{GO}(f(g, o_1, \ldots, o_m))$. 

\[\text{Proceedings of the 43rd Hawaii International Conference on System Sciences - 2010}\]
It represents a maximum preference level for each guideline $g \in G$ in states $o_1, \ldots, o_m \in \Theta$.

### 2.3. Fuzzy constraints

Let $G$ be a given set of possible guidelines which contains a solution to a decision-making problem. A fuzzy constraint $C$ is a fuzzy set on $G$ characterized by the following membership function:

$$\mu_C(g) = G \rightarrow [0,1]$$  \hspace{1cm} (7)

The function $\mu_C(g)$ establishes a maximum preference level for each guideline $g \in G$.

A set of constraints represent software development issues that developers must deal with. They represent a specific environment in where developers work. They are pieces of information extracted from the environment to contextualize the software development process.

### 2.4. Fuzzy decision

Let $G$ be a given set of possible guidelines which contains a solution to a decision-making problem. Let $GO$ be the set of fuzzy goals for the decision, represented by the membership function $\mu_{GO}(f(g,o_1,\ldots,o_m))$, with $g \in G$ and $o_1,\ldots,o_m \in \Theta$. Let $C$ be the set of constraints represented by the membership function $\mu_C(g)$, with $g \in G$.

Then, the fuzzy decision $F$ is given by the intersection of the fuzzy goals $GO$ and the fuzzy constraints $C$, represented by the following membership function $\mu_F(g,o_1,\ldots,o_m)$ with $g \in G$ and $o_1,\ldots,o_m \in \Theta$:

$$\mu_F(g,o_1,\ldots,o_m) =$$
$$= \mu_{GO}(f(g,o_1,\ldots,o_m)) \wedge \mu_C(g) =$$
$$= \min(\mu_{GO}(f(g,o_1,\ldots,o_m)),\mu_C(g))$$  \hspace{1cm} (8)

### 2.5. Optimal fuzzy decision

The optimal fuzzy decision $g^*$ is called the maximizing decision and defined by

$$g^* = \arg \max_{g \in G} \left[ \mu_F(g,o_1,\ldots,o_m) \right]$$  \hspace{1cm} (9)

with $g \in G$ and $o_1,\ldots,o_m \in \Theta$. It is the one that better satisfies the goals and the constraints. In the case of two or more guidelines with the same satisfaction level, the decision can be made by a default rule, establishing priorities between guidelines. To reduce the complexity involved in decision-making, guidelines are decomposed in sub-guidelines.

### 3. Case study on software development

During software development developers have to take several decisions related to the configuration, deployment and maintenance of software. Under this scenario, there are several guidelines representing an ideal case study to test fuzzy decision-making.

The objective of this research is to implement an algorithm to assist developers in selecting a proper set of guidelines in software development.

#### 3.1. Problem description

In this case study, a software health care application, called dicom4j, that needs to well define software configuration, to maintain software projects, and to deploy and test software is taken into consideration. This software application presents the same issues that, for our experience, have software projects during their migration into ETICS used to build, configure, deploy and test software. In particular, ETICS provides a service to help software developers but also managers and users, to better manage complexity and improve the quality of their software. During software migration to ETICS, some guidelines that should be adopted by developers have been identified. These guidelines start from two cases: 1) the deployment of the ETICS services, and 2) the development of software application. In this research only guidelines related to the second case have been considered.

#### 3.1.1. Software development guidelines

The most important guidelines are specified in the following set:

- **File software structure** that expresses the structure of the software design: it recommends putting files that are associated with a module and work together in the same directory.
- **Code software structure** that shows the structure of the software design: it recommends producing consistent, clear code and following the
conventions of the adopted programming language.

**Software portability** that expresses the possibility of using a defined platform to build a given module.

**Software dependency** that shows the necessity of avoiding conflicts and recursive dependencies inside a software application.

**Software versioning** that keeps track of different versions of a piece of software.

**Build software tool** that represents the process of compiling source code.

**Software packaging** that expresses the possibility of creating distribution packages on supported platforms in different formats.

These guidelines also match some of the steps [29], like coding and configuration management, that structure engineering processes, followed by developers of health care applications to success the software project in health care.

In literature [33], there are several kinds of structures associated with a software product. In this paper, only file and code software structures have been considered. The process of compiling source code considers several factors, such as the adoption of Version Control Systems (VCSs) to maintain their code, and of compile tools to build and test their code. The most common VCSs are SubVersion [26] and Concurrent Version System (CVS) [28, 7]. Tools to build and test code are for example ant [34] and autotool [35].

3.1.2. The set of environment states. States result from experiences performed into ETICS and literature [29, 33]. File software structure guideline specifies files as environment states, which are recommended to be included into the directory of a module. Code software structure guideline provides directories, which are recommended to be included into the directory of a module. Software dependency guideline specifies a specific version of a software package, and identifies when that version is required. Software versioning guideline provides for the environment states two different version schemas [1], in which there are the following numbers: **major** is the number increased when there are significant changes in functionality; **minor** is the number incremented when only minor features or backward compatible fixes have been added; **revision** is the number incremented when bugs are fixed; and **age** is the number incremented when the build or test process has been fixed and updated. Software packaging guideline expresses two formats of distributed packages: rpm [2] and tar [27].

According to the list of the presented guidelines the following set of states is used, grouped for guideline:

**File software structure guideline**
- **readme file** describing the module and its use;
- **status file** listing what is finished and what needs to be done;
- **install file** explaining how to install the module;
- **maintenance file** explaining how to maintain the module files;
- **license file** containing license module information.

**Code software structure guideline**
- **src directory** containing source code;
- **interface directory** containing public interface files such as files with suffixes .h or .wsdl;
- **config directory** containing configuration and scripting files such as files with suffixes .conf, .sh, and .csh.

**Software portability guideline**
- **slc4 platform**;
- **sl5 platform**.

**Software dependency guideline**
- **run time dependency** used during software installation and execution;
- **build time dependency** used during software compilation;
- **build and run time dependency** used both for compiling software and for installing and executing software;
- **version A of library B**;
- **version C of library B**.

**Software versioning guideline**
- **major, minor, revision, age** describing a version composed by four numbers;
- **major, minor** describing a version composed by two numbers.

**Build software tool guideline**
- **subversion** used as VCS;
- **cvs** used as VCS;
- **ant** for java programming language used as build tool;
- **maven** for java programming language used as build tool;
- **autotool** for the all programming languages used as build tool.

**Software packaging guideline**
- **rpm package** used on Unix operating system;
- **tar package** used on Unix operating system.

Each state is represented by a linguistic term, which provides a more natural way to express the
rating for assessing candidate guidelines. In this paper we adopt a linguistic scale consisting of seven linguistic values [36]: 1) ‘Definitely Not’ (DN), 2) ‘Probably Not’ (PN), 3) ‘Maybe Not’ (MN), 4) ‘Unknown’ (U), 5) ‘Maybe’ (M), 6) ‘Probably’ (P), and 7) ‘Definitely’ (DE). The highest linguistic value is DE, while the lowest one is ‘DN’.

3.1.3. Main goal of developers. The main goal of developers is to take as few goals as possible. This linguistic term can be defined by the membership function \( \mu(y) : R^i \rightarrow [0,1] \) from the real line \( R^i \) to the close interval \([0,1]\), where \( y \) is the number of taken goals.

3.1.4. Cost function of software development. For simplicity, only two guidelines, identified by file software structure and code software structure, are described. The algorithm is able to consider seven guidelines.

The number of taken goals depends on the guideline that is under evaluation. Each guideline is represented by one or more rules. The highest priority is given to the guideline that has the minimum value. Each rule has a cost that contributes to the total cost. The rule cost is determined by the product between its constant cost \( w \) and its activation level \( d \) given by a predicate. Each guideline has a default rule with cost 10, such that its activation level is given by

\[
\text{max} \left( 0.1 - \sum_{j=1}^{m} d_i(o_j) \right)
\]

(11)

where \( d_i(o_j) \) represents rule \( i \) activation level for guideline \( g \) operating in fuzzy state \( o_j \in \Theta \) with \( j=1,...,m \). The value of the constant cost \( w \) is defined by expertise knowledge, reflecting the answer to questions like: what average goals the software development takes if developers consider guideline \( g \) in states \( o_1,...,o_m \in \Theta \) ? or what must be the average number of taken goals when developers execute software dependency guideline in states \( o_1,...,o_m \in \Theta \) ? The expertise knowledge states that this cost must be low.

3.1.5. Constraints of software development. A maximum satisfaction level is assigned to each guideline via constraints. Examples of constraints are: coding style and making code simpler cannot be applied, therefore the code software structure guideline should not be active; dependency conflicts inside a software application cannot be avoided during software installation, therefore the software dependency guideline should not be active.

When a guideline is not activated its fuzzy value is 0, but fuzzy values for other guidelines can be set to 1.

3.1.6. Fuzzy decision-making applied to software development. The decision of what is the best guideline to select is made by evaluating all the guidelines, using the cost function and main goal function. The selected guideline has the maximum satisfaction level for the goal and the constraints.

The fuzzy operator min is used to constrain the satisfaction level of goal with the maximum level allowed by constraints.

Priorities between guidelines are expressed by the following linguistic scale consisting of five linguistic values: 1) ‘Fully Not’ (FN), 2) ‘Slightly Not’ (SN), 3) ‘Medium’ (M), 4) ‘Slightly’ (S), 5) ‘Fully’ (F). The highest linguistic value is ‘Fully’, while the lowest one is ‘Fully Not’. The following membership functions express the evaluation of the linguistic values:
where \( a, b, c, d, e \in [0,1] \), \( a < b < c < d < e \), \( a \) and \( e \) stand for the lower and upper value of the fuzzy number.

### 3.2. Guideline detection

This section describes how a given guideline is detected. Figure 2 shows the guideline selection process, which must be applied to all software development guidelines.

**Figure 2.** The decision process to select guideline.

The set of the environment states are defined by:

\[ \Theta = \{ \text{src directory DN, src directory PN, src directory MN, src directory U, src directory M, src directory P, src directory DE, interface directory DN, interface directory MN, interface directory U, interface directory M, interface directory P, interface directory DE, config directory DN, config directory MN, config directory U, config directory M, config directory P, config directory DE} \}, \]

while the set of guidelines is defined by:

\[ G = \{ \text{code software structure} \}. \]

The solution space of \( G \times \Theta \) has twenty-one elements. The mapping \( k \) associates for each guideline-states pair a consequence, like the three consequences specified below:

- if src directory, interface directory and config directory are DE and code software structure is set, the guideline code software structure is fully accepted;
- if src directory is DE, interface and config directories are U, and code software structure is set, the guideline code software structure is slightly accepted;
- if src directory, interface directory, and config directory are DN, and code software structure is set, the guideline code software structure is not accepted.

The decision function \( D \) maps the consequences to real numbers \( D_1, D_2, \) and \( D_{21} \), which impose a preference ordering on the solution set. Therefore, the best choice for code software structure guideline \( g^* \) is given when developers agree on the following states: src, interface and config directories have to be included in the software structure tree.

**Figure 3.** Graphical representation of a choice calculation for code software structure guideline: in the top the best choice is given.
Figure 4. Graphical representation of a choice calculation for file software structure guideline: in the top the best choice is given.

4. Used health care application

As health care application we have decided to use dicom4j. It is a software project with the aim to provide java source code related to the Digital Imaging and Communications in Medicine (DICOM) Standard\(^2\). The source code is in pure java making software portability easier working on all operating systems.

This project presents typical issues of software development, having to answer different needs of DICOM software developers. For example, by provisioning a DICOM framework, which implements the standards, the process of software validation can be improved. In addition, a toolkit to develop software based on this framework can facilitate the adoption of DICOM standard and the development of DICOM applications [9], like DICOM Editor to view a dicom header, Store Service Class User (SCU) to perform storage request, Worklist SCU to perform basic modality worklist request.

4.1. Adoption of guideline selection process

In the case of dicom4j, the guideline selection process has been applied to select the software portability guideline and build software tool guideline as described in Figure 5 and Figure 6 respectively. Figure 5 shows that the consensus is achieved with a medium value, when developers evaluate sl4 platform state as DE and sl5 platform state as U. Figure 6 details that the consensus is achieved with a high value, when developers evaluate subversion state as DE, maven state as DE.

**Figure 5.** The decision process to select software portability guideline.

- **Step 0:** The environment states are:
  - subversion
  - maven
- **Step 1:** Developers evaluate environment states as follows:
  - subversion DE
  - maven U
- **Step 2:** The model ranks environment states as
  - slightly
- **Step 3:** The consensus is medium
- **Step 4:** Is consensus reached?
  - Step 4.1: Yes
- **Step 5:** The guideline has been selected

**Figure 6.** The decision process to select build software tool guideline.

- **Step 0:** The environment states are:
  - slc4 platform
  - sl5 platform
- **Step 1:** Developers evaluate environment states as follows:
  - slc4 platform DE
  - sl5 platform U
- **Step 2:** The model ranks environment states as
  - fully
- **Step 3:** The consensus is high
- **Step 4:** Is consensus reached?
  - Step 4.1: Yes
- **Step 5:** The guideline has been selected

\(^2\) Digital Imaging and Communications in Medicine (DICOM), http://medical.nema.org/dicom/.
5. Simulation results

We have implemented in python the case study. Some tests were executed. The selected software development guidelines were tested considering all possible states in order to evaluate the goodness of the script itself. When the algorithm is not near the best decision for a given guideline, it decides to change guideline.

The algorithm runs in 200 µs over a Intel(R) Xeon(R) CPU E5410 @ 2.33GHz with seven guidelines. The code contains some optimisation and used python libraries that are quicker then specific code.

6. Conclusions

This research developed a fuzzy decision-making algorithm to improve software development process through a set of guidelines and to effectively solve software evaluation process. Designing rules for fuzzy decision is very intuitive, as shown in an example concerning a subset of the software development guidelines. The algorithm includes a way of representing information about the issue that has to be coped with. In addition, it constructs relations among guidelines based on the concept of fuzzy and determines ranking orders among them.

By using the algorithm, the ambiguities involved in the assessment data could be represented and processed to assure a more convincing and effective evaluation process. However, the main problem still continues to be the modus operandi of developers that usually are non-well inclined to accept changes.

References


