PiRo
SYSTEMS ENGINEERING

PiRo

an agile and holistic approach to formalize specifications

Peter Nicke
PiRo Systems Engineering GmbH
Amselweg 7, 71032 Böblingen

date of release: 10/04/2016

PiRo

SYSTEMS ENGINEERING

Contents

1. Introduction

Today development projects are getting more and more complex. Complex in terms of software structure, amount of functionality, number of stakeholders and especially in terms of the amount of software communication partners (e.g. other software modules, ECUs and systems).

To handle this complexity a couple of measures were introduced. For example:

- agile methods for project management (e.g. SCRUM)
- software platform architectures and design rules (e.g. AUTOSAR in the automotive industry)
- .....

Nevertheless, only 14% of all the IT projects are successful. 57% are critical leading to big time and budget overrun and 29% are failing completely (consolidated numbers 2002-2010<5>). The three most important reasons for an unsuccessful project are:

1. missing preliminary work of stakeholders and users
2. incomplete or unclear requirements

3. frequently changing requirements

One of the most important but mostly undervalued part of development is requirements engineering. Clear requirements are one of the 3 most important criteria for a successful project<5>.
A NASA study shows, that with an increased requirements engineering budget of 5-10% of the overall project budget the cost overrun was diminished from 150% to 25%<7>.

Definition of Requirements Engineering according to IREB<4>

The requirements engineering is a systematic and disciplined approach for specification and management of requirements with the following objectives:

- know the relevant requirements, get consensus between the stakeholders, document the requirements in compliance to given standards and manage requirements systematically
- understand and document the stakeholders wishes
- specify and document the requirements to minimize the risk to deliver a system which not fulfills the stakeholders wishes and needs

Without proper system objectives, system and context boundaries with corresponding interfaces and requirements, nobody knows exactly how to implement the product. Experiences have shown that a lot of time and money are wasted caused by wrong implementations due to bad or missing requirements. In IT projects 60% of all failures are caused by analysis failures. Whereas only 40% are referable to implementation issues<3>.

Once the requirements are available it comes down to test specifications to confirm the described functionality. Experience shows that often a huge amount of man power is spend but the product still shows wrong behavior which was not discovered by the tests. Such problems often lead to extra bug fixing releases or cost expensive postponing of product releases.
Insufficient test results are caused by various reasons like underestimated test know how or missing test platforms and test equipment.

But one of the major problem is based on bad test case definitions. Test cases often do not represent the corresponding requirement in correct form. Without a suitable test case definition it is not possible to get satisfying test results.

Today most of the requirements and/or test cases are either written with

- proses (natural language)
- complex description languages (UML, CafeObj, CASL, TTCN-3 ...).

Prose leads often to ambiguous descriptions with doubtful results whereas the several description languages are quite difficult to understand for non-professionals.
With prose it is also not possible to get automated test scripts for HiL and other applications. This leads to additional workload to generate automated test scripts.

PIRO

PIRO is a solution which combines the legibility with machine readability.
PIRO uses the approach of keyword driven test cases<2>.
PIRO defines a simple syntax and unambiguous semantic by using predefined phrases which are close to our natural language.

PIRO *is a new agile approach to express each specification as abstractly as possible while making it precise enough to interpret and execute it automatically.*

The legibility can only be achieved by using a big number of predefined phrases like it is known by a natural language (instead of a very few instructions like known from different modeling languages). Therefore, a new approach by using a "living" database with phrases is introduced. Every stakeholder is able to define new phrases.
As PIRO provides a specific syntax and semantic, an automatic translation in different languages (natural language, HiL scripts, ...) is possible. The dictionary is also part of the database.

1.1 Motivation

The idea of PIRO was built based on test case specifications. The daily experience with persons who were writing test cases and persons using them to write HiL scripts have shown a big amount of unnecessary and inefficient communication effort. Especially when it comes to different cultures and different development locations (with sometimes different time zones). Observations have shown a couple of causing issues:

1. Test cases are not precise and often ambiguous
2. A lot of unnecessary information was used
3. Different languages
4. "Chinese whispers issue" [1.a]
5. Implicit knowledge

Those problems can be avoided by standardizations. Out of this PIRO was born.

1.2 Visions and advantages

The following premises were crucial to get the standardization. The enumeration is sorted by priority

1. User friendly
   1.1 legibility
   1.2 intuitively useable, especially for new users
2. Machine readability
3. Unified grammar (syntax and semantics) from requirement to test case
4. Fast and correct method to get specifications written

[1.a] Change of content on the way from requirement to test case to test script

5. Independence of

    5.1 Specification and test platforms (DOORs, DanTe, ALM, ...)

    5.2 Phases of the concrete development cycle (e.g. valid for all levels of the V-model)

6. Scalability

    6.1 Type of user (customers, suppliers, service partner, ...)

    6.2 Arbitrary independent projects

With the machine readability the following features can also be realized:

- translation in nearly arbitrary languages and test platforms
- tool supported requirement and test case definition

Other advantages which result out of the described implementation are:

- The approach of keyword driven testing leads to the possibility to start writing test cases without a corresponding test system implementation (e.g. HiL).
- With the automated translation into HiL test scripts it is possible to define a unique structure, labeling and annotation.
- A consistent traceability between test case and HiL implementation can be realized.
- The requirements and test cases get a measurable quality level. A proper metric can be defined without the necessity of having results in the form of implementation failures like it is required in existing metrics.
- traceability between requirement and test case allow an automatic consistency checkup

2. Basic concepts and elements of PiRo

PIRO provides 3 different mechanisms to get the best specifications in class.

- PiRo::Phrases
- PiRo::Actions
- PiRo::RequirementTemplates

The most important elements are the PiRo::Phrases. The Actions and Requirements Templates complete PIRO to a holistic approach which fulfills the ISO/IEC/IEEE-29148<1> quality characteristics for requirements.

In the simplest case a PiRo::Phrase represents a parameter, a timer or another object. But a PIRO phrase can also cover complex functions, activities or signal definition which may be composed from other PIRO phrases. PiRo::phrases are used by domain experts to assemble PIRO requirements or PIRO test cases.
Using PIRO phrases brings the following advantages:

Reusability and Single Source

Often complex definitions are used multiple times within a specification. With PIRO the definition can be done in one particular place. Within the corresponding requirements the predefined phrase is used. This makes the work with specifications much more efficient.

Clear specifications

Less words are making sentences easier to read! Sentences are optimal legible with less than 9 words<6>. If only a small phrase is used instead of a complex definition the specification get much more comprehensibly.

Easy Change Management 1

Often parameters, timers, signals etc. are used for more than one requirement or test case. With the use of PIRO phrases the change is minimally invasive as only one particular requirement has to be changed instead of all the corresponding requirements or test cases.

Easy Change Management 2

One of the major quality problems in IT projects is the high frequency of requirements changes. This often leads to changed requirements on system level but the necessary changes on software level or within the corresponding test specifications are missing or delayed. As the changes often only affect simple signal and parameter definitions an automatic and prompt translation with PIRO phrases can solve such issues.

UML Action Languages

Today the model based testing becomes more and more important. Therefore, a model and its artifacts has to be described by triggers, guards, effects and different other parameters (depending on the diagram type). When it comes to the automation well defined expressions are needed. PIRO can also be used to define those artifacts.

Domain invariance

Test specifications can be written without the need to do this domain specific (test platform specific). Afterwards the domain independent test specification can easily be translated to specific test platforms by using a PIRO dictionary where the phrases get translated domain specific. The number of needed test cases can be reduced dramatically.

Traceability

A big problem for software quality is the traceability between requirements, test specifications and test scripts. With a responsible requirements engineer the different specifications are linked together. But this does not ensure the correspondence of the content. By using PIRO phrases and an automatic translation it can be ensured that the same behavior (e.g. signal definition) is used for every specification.

Runtime enhancement

Amount of test cases may also be consolidated by an automatic synergy check of all automatically generated test cases

2.1 PiRo Test Cases

As the most of the test case documents are divided by

- precondition
- action
- expectation

or a similar structure, there is no need for a requirement template. A verb is optional and can be used to improve the flow of reading. Therefore, test specifications can be handled only by using PIRO phrases. The following example shows a simple test case to test if a car drives backwards when a reverse gear is engaged.

Table 2.1: PIRO test case

TextCase ID	precondition	action	expectation
PiRo-0815		1) car::engine[running] 2) GearShift[R] 3) AccelPedal[pressed]	DrivingDirection[backward]

With a proper database the test case can automatically be translated into the different test platform specific languages. Besides the particular HiL systems with their specific domain language a translation is also possible into driver instructions (table 2.2) or bus signals (table 2.3):

Table 2.2: PIRO test case translated into driver instructions

TextCase ID	precondition	action	expectation
DI-0815		1) start engine 2) engage reverse gear 3) press accelerater pedal	The car drives backward

Table 2.3: PIRO test case translated into bus signal (signals are not real)

TextCase ID	precondition	action	expectation
DI-0815		1) EngineStat[running] 2) GearShift_Target[R] 3) AccelPedalWay[>10]	RollingDirect_FL[Back]

2.2 PiRo Requirements

On the contrary to the test specifications, requirements often do not have a predefined structure. Therefore, the PiRo::RequirementsTemplate and a PIRO verb is used. A very simple proposal: It is required that a car drives backward if the engine is running, a reverse gear is engaged and the acceleration pedal is pressed.

Table 2.4: PIRO Requirements

Req ID	Requirement	Comp

Car-2016	If **car::engine[running]** AND
	if **GearShift[R]** AND
	if **AccelPedal[pressed]**, the system shall **drive DrivingDirection[backward]**
		car

Here the PIRO verb "drive" and the PIRO phrases "car::engine[running]", "GearShift[R]" and "AccelPedal[pressed]" were used. As the requirement is functional, it can automatically be translated into a PIRO test case and afterward into different domain specific test languages.

2.3 PiRo Action Language

PiRo can also be used to get a well defined Action Language. This is the first step to automatically test State Charts. This chapter is not final!

PiRo
SYSTEMS ENGINEERING

3. PiRo Syntax

3.1 The PiRo::Phrase

PiRo uses predefined keywords to define the phrases. Such a phrase is a compound string without any blanks. To get a phrase still easy legible a proper syntax is needed. It is recommended to use the CamelCase method[3.a] or the separation by an underscore to define a PiRo::Phrase.

In general a PiRo phrase describes an entity[3.b] whereby an entity is at least characterized by its name.

PiRo - *Definition: minimal* PiRo *-Phrase*
The minimal phrase is given by the entity itself.

For example if the specification often refers to an particular "wake up time" the following phrase can be defined to get the benefits mentioned above:

WakeUpTime

3.1.1 The Phrases Class

To get the specification structured and easy to read often the class of the entity (the object type) is used. In this case the *Entity* is prefixed by its *Class* whereby they are separated by two double dots.

[3.a] CamelCase (also camel caps or medial capitals) is the practice of writing compound words or phrases such that each word or abbreviation begins with a capital letter. Camel case may start with a capital or, especially in programming languages, with a lowercase letter.

[3.b] An entity is something that exists in itself, actually or potentially, concretely or abstractly, physically or not. But there is no need of material existence. In particular, abstractions and legal fictions are usually regarded as entities

PIRO - *Definition: class of the* PIRO *-Phrase*

class::Entity

For example if a parameter "wake up time" should be described, the following PIRO -Phrase can be used:

param::WakeUpTime

With the *Class* it is also possible to define the characteristics of an entity. For example if the parameter "wake up time" should be implemented as SCN codable parameter the following phrase can be used:

SCN::WakeUpTime

With a corresponding requirement that all entities which are defined in class "SCN" have to be implemented as SCN parameter within the software.
A list of recommended classes can be found at Page 32.

3.1.2 The Phrases Argument

If it is needed to define a concrete *Argument* for an entity this can be done in the following form:

PIRO - *Definition: Argument of the* PIRO *-Phrase*

entity[argument]

The *Argument* is optional.
For example if the SCN parameter "wake up time" should be set to "1000", the following PIRO -Phrase can be used:

WakeUpTime[1000]

- It is allowed to use more than two ore more arguments combined by logical operators and relations
  - AND
  - OR
  - NOT
  - <,>,=,>=, >=;
- If a flank from one state to another should be described the conjunction "TO" is used (e.g. **itf_FR::Ignition[IGN_ON** TO **IGN_START]**)

3.1.3 The Phrases System and Subsystem

Often the PIRO phrase is not unique or it is not clear in which component the entity is handled. Therefore, it can be extended by the systems in which the entity exists.

System and Subssystem of the PIRO *-Phrase*

system::subsystem::entity

For example if the SCN parameter "wake up time" refers to the system "ESP", the following PIRO -Phrase can be used:

ESP::WakeUpTime

And if this is still not sufficient the following proposal is also possible:

Mercedes::ESP::WakeUpTime

In principle it is allowed to define the phrase with more than two systems. But as the legibility was one of the major PIRO objectives it is not recommended.

3.1.4 The maximum Phrase

All the described elements can be combined together in the following form:

PIRO - *Definition: maximum* PIRO *-Phrase*

class::system::subsystem::entity[argument]

For example if the SCN parameter "wake up time" of the system "ESP" within the mercedes car should be set to "1000", the following PIRO -Phrase can be used:

SCN::Mercedes::ESP::WakeUpTime[1000]

It is important to use the class at the beginning of the phrase to improve the legibility.

3.2 PiRo::Action

Once some PIRO phrases are defined the question will come up how to handle them to get a legibly requirement. Therefore PIRO introduces Actions (What verbs are in a sentence that are PIRO actions in the specification). PIRO is able to handle free defined Actions like known by the phrases.
Mostly Actions are not needed to define a required behavior as they are implicit given by the particular phrase. But it is urgently recommended to use them because:

- The requirement get only legible with Actions / Verbs like known by a natural language
- The driver instructions which are potentially translated out of requirements get comprehensible

For example if the FlexRay signal "GearShift" should be set to a particular signal value (for example to "R") the Action "set" can be used:

set itf_FR::GearShift[R]

Other recommended Actions are "show" (for GUI relevant Phrases), "drive", "request" and a lot more.

3.2.1 Wait[]

When it comes to test cases and relevant timing definitions often a well defined time is necessary to specify the exact behavior.
For example it is often required to define a time between two inputs or to wait for an particular event before starting a new action. This can be done by using the "wait[]" Action. The argument can either be a timing definition or an external event which should be described by a PIRO phrase.

Timing definition

If the tester should wait some milliseconds between two actions, the "wait[]" action can be used in the following form:

TextCase ID	preconditio	action	expectation
PiRo-0815		1) car::engine[running] 2) GearShift[R] 3) **wait[2000]** 4) AccelPedal[pressed]	DrivingDirection[backward]

The default unit are milliseconds.

Event

If the tester should wait for an event between to actions, the "wait[]" action can be used in the following form:

TextCase ID	preconditio	action	expectation
PiRo-0815		1) car::engine[running] 2) GearShift[R] 3) **wait[until SVSView[TV+RV]]** 4) AccelPedal[pressed]	DrivingDirection[backward]

In this example the tester should wait till the camera view "TopView+RearView" is shown before the accelerator pedal is pressed.

3.3 PiRo::RequirementsTemplate

With the PiRo phrases and verbs all components are available to create a complete and comprehensible requirement. The PiRo requirement template is based on the IREB standard<4>.

If a simple function is required the following template should be used

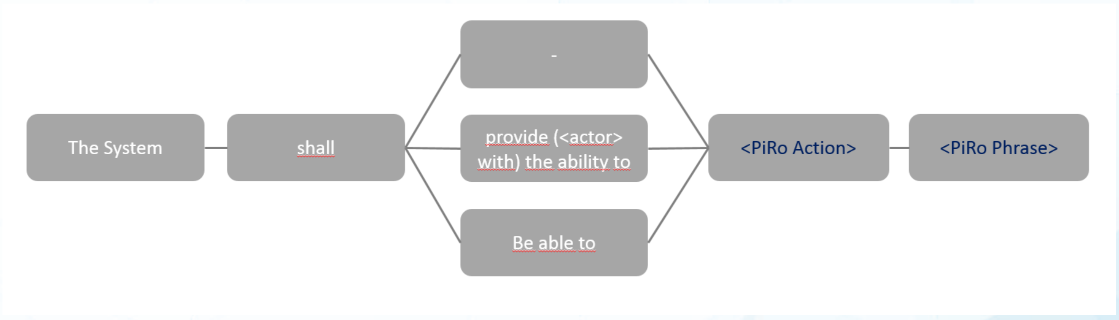

Figure 3.1: RequirementTemplate "Funktion"

If a function is triggered by an event or a particular status, the following template should be used:

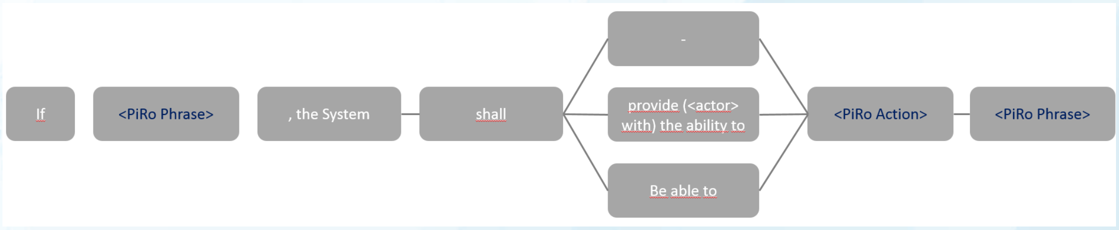

Figure 3.2: RequirementTemplate "Condition"

If more than one condition is necessary they can be combined by "AND" with another "if" per condition.

To get a high quality requirement the following recommendations should be considered:

- use "the system" instead of the particular component name as it is defined by a particular attribute anyway

- highlight the PIRO verbs and phrases by bold text to declare them as "well defined"

4. PiRo Semantic und Translation

4.1 The PiRo Phrase Content

Next to the advantages mentioned in chapter "PiRo Syntax" on page 8 the main functionality of PiRo is the translation functionality from the PiRo::phrase to other domain specific languages. Therefore, a proper database is needed to define the PiRo phrase and its translation. It is also recommended to use comments, keywords and other proper descriptions which potentially can support other tool functionality.

1. phrase
2. description of phrase class
3. description of phrase system/subsystem/...
4. description of phrase Entity
5. examples
6. keywords
7. translation tables
   - 7.1 driver instructions
   - 7.2 HiL scripts
   - 7.3 bus signals
   - 7.4 ....

4.2 Translation To Other Languages

We distinguish between two translations

1. PiRo Requirement to PiRo Test Case
2. PiRo Test Case to test platform specific languages (different HiL scripts, driver instructions, bus signals definitions, ...)

An automatic translation from requirements to any other language is only possible for well defined functional requirements. In case of non functional requirements a translation is only partly possible. Nevertheless also for non functional requirements the engineer benefits due to the fact that the test case is partly given.

5. ISO/IEC/IEEE 29119 conformity

End of 2015 the ISO/IEC/IEEE 29119 Part 5 was published. The standard describes the Keyword-Driven Testing and discusses the following topics:

- Introduction in Keyword-Driven Testing
- Application of Keyword-Driven Testing
- Roles and Tasks
- Frameworks for Keyword-Driven Testing
- Data Interchange

PIRO covers the proposed syntax and extends it for requirements. With new approaches like namespaces, systems and classes (the standard only propose to have arguments) PIRO is able to handle any kind of specification and provides a holistic method for requirement engineering and test management.
PIRO does not distinguish between low and high level keywords. The motivation is not clear and from PIRO point of perspective there is no need to do this.
PIRO uses only one abstraction layer because at the moment it is bespoken on system and software acceptance tests. This is proposed by the standard.

PiRo is compliant with the standard and extends the proposed mechanism. Some parts like the keyword library are not discussed yet but will be prepared asap.

6. ISO/IEC/IEEE 29148-2011 conformity

The ISO/IEC/IEEE 29148-2011 discusses the entire system and software engineering topics. As PIRO supports the requirement engineering, only this part of the standard was consulted. Especially the quality criteria for requirements are considered. The standard proposes the following characteristics:

1. Necessary
2. Implementation free
3. Feasible
4. Singular
5. Traceable
6. Verifiable
7. Complete
8. Consistent

9. Unambiguous

Especially the last 5 characteristics are supported by PIRO .

PIRO is compliant with the standard.

7. Outlook

7.1 Parameter Usage

Motivation: usage of loops to vary values in test cases

This chapter is not finalized!

7.2 Other Requirements templates

7.3 Automatic Translation of UML diagrams

A. Appendix

PiRo
SYSTEMS ENGINEERING

A.1 Recommended PiRo Classes

A.1.0.1 General

- param::
- timer::

Every interface of the component to specify should be adressed. For example:

- itf_LVDS:: (for a Low Voltage Differential Signal interface)
- itf_FR:: (for a FlexRay interface)
- itf_CAN:: (for a CAN interface)
- ...

A.1.0.2 UML

For UML the following standard systems are proposed:

State Chart

- state::
- transition::

Class Diagramm

- class::
- aggregation::
- composition::
- generalization::
- ....

A.1.0.3 ECUs in automotive environment

- SCN::
- EVC::

PiRo

SYSTEMS ENGINEERING

List of Figures

PiRo

SYSTEMS ENGINEERING

Bibliography

[1] 1, I. J.: IEEE 29148: Systems and software engineering. Life cycle processes. Requirements engineering. In: *IEEE* (2011)

[2] 1, I. J.: IEEE 29119: Software and systems engineering - Software testing. In: *IEEE* (2013)

[3] BOEHM, B. : *Software Engineering Economics*. Englewood Cliffs, Prentice-Hall, 1981

[4] CHRIS RUPP, K. K.: *Basiswissen Requirements Engineering*. 4. dpunkt.verlag GmbH, 2015

[5] GROUP standish: CHAOS report. (2011)

[6] SCHNEIDER, W. : *Deutsch für Profis: Wege zu gutem Stil*. 14. Goldmann, Munich, 2015

[7] YOUNG, R. R.: *Effective Requirements Practices*. Addison-Wesley Longman; Amsterdamm, 2001

PiRo

SYSTEMS ENGINEERING

Index