



Requirements Management Tool rmtoo

Details

by flonatel GmbH & Co. KG

Version 3

Content

1. Audience
2. Requirements
3. Topics
4. Configuration
5. Processing
6. Output Artifacts
7. Further Reading
8. Future

1. Audience

Audience

- Anybody who is interested in open source requirements management tools.
- Anybody looking for a good and usable requirements management tool.
- Prerequisite for this presentation: basic knowledge of rmtoo – as described in *rmtoo – Overview and Feature Set*

2. Requirements

Requirements: Major Attrs

- Requirements have a fixed set of attributes
- Major attributes:
 - Name: headline of the requirement
 - Description: content of the requirement
 - Id: unique id used for referencing
 - Depends on: / Solved by: create dependencies between requirements
 - Topic: assignment to topic

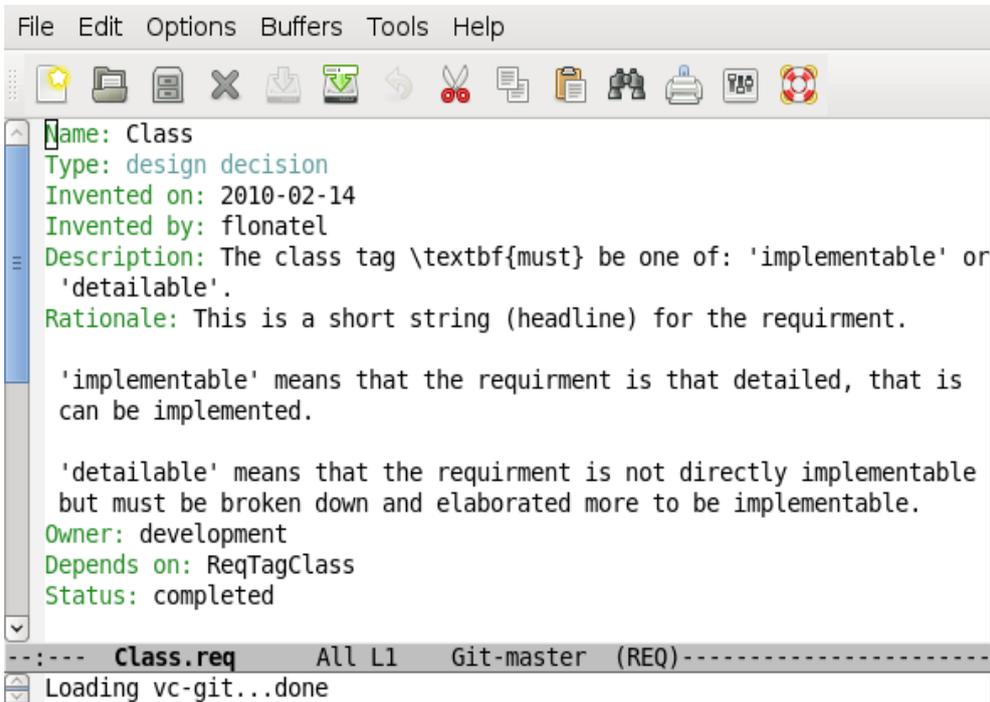
Requirements: Minor Attrs

- Minor attributes:
 - Rational: and Note: details of the description and further information
 - Invented on / by and Owner
 - Type: requirement or design decision
 - Status: finished or not done
 - Priority / Effort Estimation / Class: handling of agile development approach

Input Files

- Input files are plain text files
- Each requirement is noted as a list of key-value pairs
- Each attribute is mapped to a tag
- Files can be handled by most *nix commands (sed, streplace, awk, grep, ...)
- Version control can be carried out by version control system

Input File Example



```
File Edit Options Buffers Tools Help
[Icons]
Name: Class
Type: design decision
Invented on: 2010-02-14
Invented by: flonatel
Description: The class tag \textbf{must} be one of: 'implementable' or
'detailable'.
Rationale: This is a short string (headline) for the requirement.

'implementable' means that the requirement is that detailed, that is
can be implemented.

'detailable' means that the requirement is not directly implementable
but must be broken down and elaborated more to be implementable.
Owner: development
Depends on: ReqTagClass
Status: completed
--:-- Class.req All L1 Git-master (REQ)-----
Loading vc-git...done
```

- Simple *key: value* notation
- Space in first column: extend value
- Keys are fixed (predefined)
- Values are checked whenever possible
- Editable with standard text editor

3. Topics

Topics

- Topics are the way requirements are organized
- They provide the structure of the output document(s)
- Each requirement must be assigned to a topic

The Idea behind Topics

- Each topic represents one chapter
- Name is the headline
- Text is additional text
- Subtopics are sub-chapters / sections
- Requirements which are assigned to a topic are printed in the corresponding chapter

Topic Attributes

- A topic contains
 - Name: a headline
 - Text: arbitrary text for explanations
 - IncludeRequirements: all requirements contained in the topic
 - Subtopic: other topics
- As many subtopics and texts can be included as necessary

Input Files

- Input file format is generally exactly the same as the input file format for requirements.

Input File Example

- Same format as the requirements input files

```
File Edit Options Buffers Tools Help
Name: rmt00
SubTopic: Introduction
Text: This is the \textsl{master} requirment where all other
      (especially the initial requirements) depend on. There can only be
      one master requirment.
IncludeRequirements: full
SubTopic: Basics
SubTopic: ReqTags
SubTopic: Topic
SubTopic: Input
SubTopic: Output
SubTopic: EmacsMode
SubTopic: Documentation
SubTopic: AutoCreationOfArtifacts
SubTopic: Testing
SubTopic: Deployment
----- ReqsDocument.tic All L1 Git-master (Fundamental) -----
Loading vc-git... done
```

4. Configuration

Configuration

- Configuration file includes details about
 - Input requirements: location and version
 - Input topic sets (multiple possible)
 - Processing: checks to perform
 - Output artifacts: configuration of all aspects for different output formats
 - Stakeholder
 - Authors

5. Processing

Processing

- rmtoo reads in
 - Requirements files
 - Topic files
 - Configuration file
- Parses and checks all input files
- Creates the configured output artifacts

Checking Requirements

- Consistency checks are carried out
- Checks include:
 - Syntax / formatting checks e.g. for date fields
 - Type checks: some fields are allowed to contain only a limited set of (key)words
 - Typo checks e.g. for owners
 - Dependency checks
- Problems are directly reported

Example: Checks

```
Defects.req
File Edit Options Buffers Tools Help
Name: Defect Requirements Description
Status: not able to implement
Invented on: 2010a-02-14
Invented by: flonatel
Descriptio: A text file showing some checks on requirements.
Priority: developmnt:10
Owner: mnanment
Type: requirment
```

```
florath@kreon: ~/devel/rmtoo
florath@kreon:~/devel/rmtoo$
florath@kreon:~/devel/rmtoo$
florath@kreon:~/devel/rmtoo$ make
./bin/rmtoo -m . -f doc/requirements/Config.py \
-d doc/requirements -c dot -o reqtree.dot -l doc/latex
+++ ERROR Defects: Status tag invalid 'not able to implement'
+++ ERROR Defects: invalid date specified (must be YYYY-MM-DD) was '2010a-02-14'
+++ ERROR Defects: stakeholder 'developmnt' not known
+++ ERROR Defects: invalid owner 'mnanment'. Must be one of the stakeholder ['development', 'users', 'customers']
+++ ERROR Defects: does not contain the tag 'Description'
+++ ERROR Defects: invalid type field 'requirment': must be one of ['master requirement', 'initial requirement', 'design decision', 'requirement']
```

Requirements' Quality

- *rmtoo* supports some heuristics to check the quality of requirements
- Each heuristic evaluates the requirement and adds or subtracts a number of points
- If the resulting point count is negative a warning is generated

Heuristics

- Good requirements
 - contain only one short sentence
 - use words like *must*, *exactly*
- Bad requirements
 - use enumerations
 - use words like *may*, *possible*, *and*, *or*

Output Artifacts

- If all checks succeed, the configured artifacts are created.
- It is possible to have a whole set of different output documents – based on different topics.
- All different output artifacts are handled during a single run of *rmtoo*.

6. Output Artifacts

Requirements Document – Requirements
Dependency Graph – Project Backlog – Project
Elaboration List – Requirements Count Stats –
Commercial Pricing

Output: Requirements Document

- *rmtoo* can create a requirements document containing all requirements
- Intermediate output format of requirements is LaTeX using hyperref
- Resulting documents can be PDF and HTML for example
- Links in table of contents and dependencies available in PDF and HTML
- Arbitrary text can be added

Output: Table of Contents (PDF)

Contents

1 Status	5
1.1 Backlog	5
1.2 Requirments Elaboration	5
2 What's all about	5
2.1 rntoo	5
3 Initial Requirements	6
3.1 rntoo must work on Requirments	6
3.2 Agile Development Process	6
3.3 Eighty Percent Rule	6
3.4 Open Source rntoo	7
3.5 Easy Extensible	7
3.6 Automatic Generation of Results	8
3.7 Easy Editable	8
3.8 Documentation	8
4 Requirements Tags	9
4.1 Requirements Name	9
4.2 Requirements Type	9
4.3 Requirements Invented By	9
4.4 Requirements Invented On	10
4.5 Requirements Description	10
4.6 Requirements Owner	10
4.7 Requirements Status	11
4.8 Status	11
4.9 Requirement Priority	12
4.10 Priority Format	12
4.11 Requirments Class	12
4.12 Class	13
5 Implementation Decisions	13
5.1 Use Txt	13
5.2 Use Python	13
5.3 Traceability	14

- Each requirement fits into it's own subsection
- Hyperlinks for fast navigation

Output: Requirement (PDF)

4.3 Requirements Invented By

Description: Each requirement **must** have a 'invented by' tag.

Rationale: This names the original (initial) author of the requirement.

Note: None

Depends on: [3.1 rmtoo must work on Requirments](#),

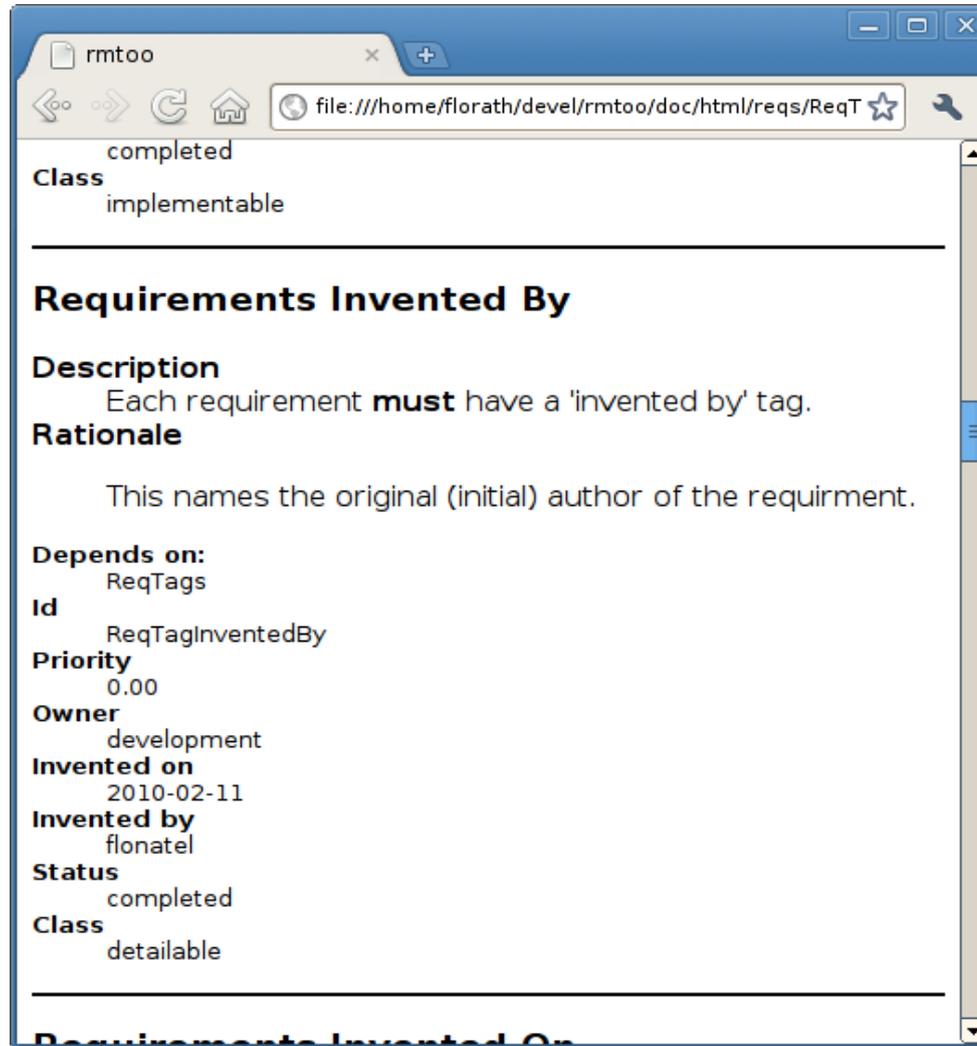
Id:	ReqTagInventedBy	Priority:	0.00	Owner:	development
Invented on:	2010-02-11	Invented by:	flonatel	Status:	completed
Class:	detailable				

- Each requirement fits into it's own subsection

All input key-values are available

- Hyperlinks to dependencies for fast navigation

Output: HTML Output



6. Output Artifacts

Requirements Document – Requirements
Dependency Graph – Project Backlog – Project
Elaboration List – Requirements Count Stats –
Commercial Pricing

Output: Dependency Graph

- *rmtoo* can create a requirement dependency graph
- Visualize requirements dependencies
- Color coded status information. For example: red font means open, black font means completed

6. Output Artifacts

Requirements Document – Requirements
Dependency Graph – Project Backlog – Project
Elaboration List – Requirements Count Stats –
Commercial Pricing

Output: Project Backlog

- *rmtoo* can create the project backlog as used in SCRUM
- The project backlog contains all the requirements which have been elaborated upon (which means they can be implemented)
- The project backlog is the ToDo list for the developers
- Requirements are sorted by priority

Output: Project Backlog Example

1.1 Backlog

Prio	Chap	Requirement Id	EfE	Sum
10.00	12.2	Test Before Packaging	3	3
2.40	11.1	Makefile	5	8
2.06	9.6	Man Page Artifact Elaboration List	3	11
2.06	9.7	Man Page Requirements Dependency Graph	3	14
2.06	9.4	Man Page LaTeX Output	3	17
2.06	9.8	Man Page Emacs Mode	3	20
2.06	9.9	Man Page Artifact Backlog	3	23
1.68	11.2	Makefile Dependencies	8	31

- Prioritized list of requirements
- Hyperlinks for fast navigation
- Effort estimation included
- Embedded in the PDF / HTML document

6. Output Artifacts

Requirements Document – Requirements
Dependency Graph – Project Backlog – Project
Elaboration List – Requirements Count Stats –
Commercial Pricing

Output: Project Elaboration List

- *rmtoo* can create a list of all requirements that must be further elaborated upon
- The Elaboration List is the ToDo list for the SCRUM master
- Requirements are sorted by priority

Output: Project Elaboration List Example

1.2 Requirements Elaboration List

Prio	Chap	Requirement Id	EfE	Sum
10.00	3.1	rmtoo must work on Requirments	3	3
10.00	2.1	rmtoo	5	8
10.00	12.3	Test Integration	13	21
10.00	12.5	Test Tool: python-nose	5	26
10.00	3.9	Ease of Use	3	29
10.00	12.1	rmtoo Automated Testing	3	32
10.00	13.1	Packaging	3	35
10.00	12.4	Unit Testing	13	48
9.00	3.3	Simplicity	21	69
8.10	5.1	Checks	8	77
6.30	8.1	Emacs Mode	8	85
6.00	3.5	Easy Extensible	5	90
5.50	3.8	Documentation	3	93
5.00	3.2	Agile Development Process	5	98
4.41	8.2	Emace Mode to Support Traceability	3	101
4.12	9.1	Documentation Man Page	5	106
3.78	8.7	Emace Mode Value Highlighting	3	109
3.00	3.6	Automatic Generation of Results	3	112
3.00	3.7	Easy Editable	5	117
2.00	7.1	Output of Different Artifacts	8	125
1.60	7.4	Output of Text Document	8	133
1.50	6.3	Traceability	13	146
1.28	7.8	Text Base Description Choose Base Tags	13	159
1.28	7.7	Output of Text Document Use Same Base	13	172
1.28	7.9	Text Base Description Requirement References	13	185
1.28	7.5	Output of HTML	13	198
1.28	7.3	Output of PDF	13	211

- Prioritized list of requirements
- Hyperlinks for fast navigation
- Effort estimation included
- Embedded in the PDF / HTML document

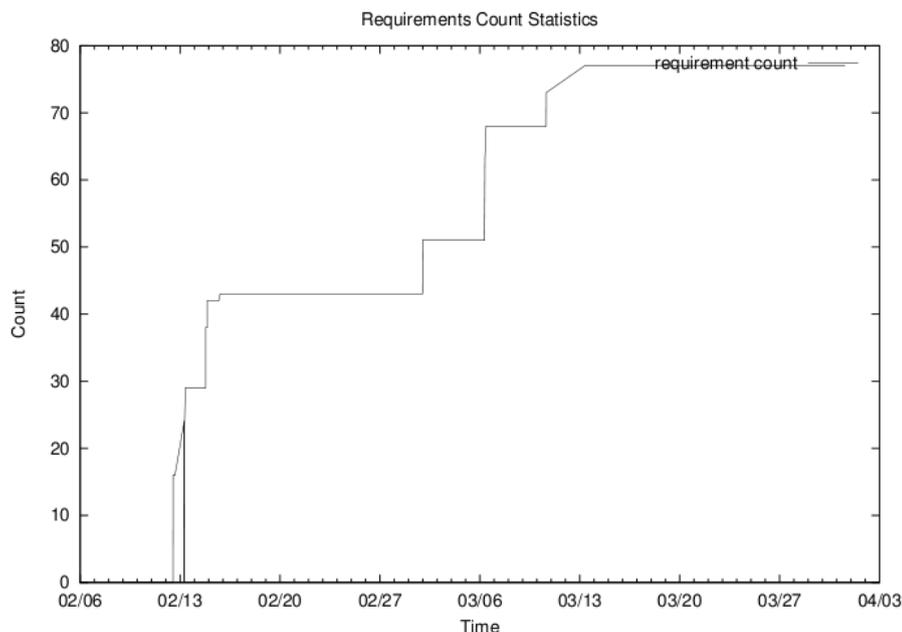
6. Output Artifacts

Requirements Document – Requirements
Dependency Graph – Project Backlog – Project
Elaboration List – Requirements Count Stats –
Commercial Pricing

Output: Requirements Count Stats

- *rmtoo* can create a cvs file for the whole history of the number of requirements
- Prerequisite: the history must available in a git repository
- Can be preprocessed using spreadsheet or gnuplot

Output: Requirements Count Stats Example



- Number of requirements at a given point in time
- Embedded in the PDF / HTML document

6. Output Artifacts

Requirements Document – Requirements
Dependency Graph – Project Backlog – Project
Elaboration List – Requirements Count Stats –
Commercial Pricing

Idea: Commercial Pricing

- *rmtoo* is able to create a spreadsheet document for pricing
- Whether for an entire project or parts of a project, a set of documentation can be created which includes precisely the information needed
- Sets of documents can be sent to vendor
- Vendor fills in pricing
- *rmtoo* reads in and evaluates results

Document: Commercial Pricing

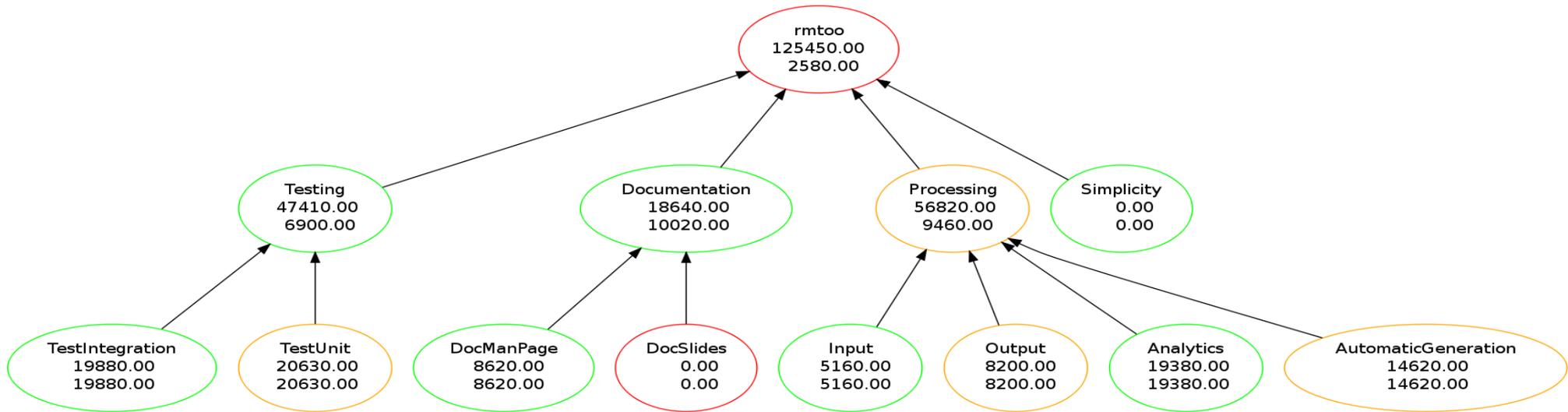
- Each requirement can be separately priced
- The price for each requirement is the sum
 - Of the requirement itself, and
 - All dependent requirements
- Current implementation software project centered – can be easily adapted

Example: Commercial Pricing

Id	Name	Compliance	Costs for requirement			Dependent from	Costs of dependent			Overall sum			Supplier	
			dayrate	#days	material sum		rate	material	sum	rate	material	sum		
rmtoo	rmtoo	none	860 €	3	0 €	2,580 €		96,700 €	26,170 €	122,870 €	99,280 €	26,170 €	125,450 €	
Simplicity	Simplicity	fully	0 €	0	0 €	0 €	rmtoo	0 €	0 €	0 €	0 €	0 €	0 €	
Testing	rmtoo Automated Testing	fully	750 €	7	1,650 €	6,900 €	rmtoo	24,750 €	15,760 €	40,510 €	30,000 €	17,410 €	47,410 €	
Processing	Processing	partial	860 €	11	0 €	9,460 €	rmtoo	46,440 €	920 €	47,360 €	55,900 €	920 €	56,820 €	
Automatic Generation	Automatic Generation of Results	partial	860 €	17	0 €	14,620 €	Processing	0 €	0 €	0 €	14,620 €	0 €	14,620 €	
Documentation	Documentation	fully	540 €	12	3,540 €	10,020 €	rmtoo	4,320 €	4,300 €	8,620 €	10,800 €	7,840 €	18,640 €	
DocSlides	Documentation Slides	none	0 €	0	0 €	0 €	Documentation	0 €	0 €	0 €	0 €	0 €	0 €	unclear
Analytics	Analytics	fully	860 €	22	460 €	19,380 €	Processing	0 €	0 €	0 €	18,920 €	460 €	19,380 €	
Output	Output of Different Artifacts	partial	860 €	9	460 €	8,200 €	Processing	0 €	0 €	0 €	7,740 €	460 €	8,200 €	
TestUnit	Unit Testing	partial	750 €	17	7,880 €	20,630 €	Testing	0 €	0 €	0 €	12,750 €	7,880 €	20,630 €	
Input	Different Inputs	fully	860 €	6	0 €	5,160 €	Processing	0 €	0 €	0 €	5,160 €	0 €	5,160 €	
DocManPage	Documentation Man Page	fully	540 €	8	4,300 €	8,620 €	Documentation	0 €	0 €	0 €	4,320 €	4,300 €	8,620 €	
TestIntegration	Test Integration	fully	750 €	16	7,880 €	19,880 €	Testing	0 €	0 €	0 €	12,000 €	7,880 €	19,880 €	

- Automatic dependency price sum computation
- Vendor is able to pick the most suitable dependency

Evaluation



- rmtoo reads in the data from the spreadsheet and evaluates it
- Upper number: complete price of subtree
- Red: not compliant – Green: compliant

7. Further Reading

Documentation

- *rmtoo* comes with a complete set of documents
- Readme.txt and FAQ.txt as next steps of reading
- About 20 man pages: introduction and reference to all man pages can be found in `rmtoo(7)`
- Complete requirements for *rmtoo* itself

8. Future

Output Modules

- An increasing number of *rmtoo* users ask for extensions of existing or complete new output modules
- Easy to write – easy to adapt
- Current complexity range
 - version1 has 31 lines of code
 - ooprising1 has 726 lines of code

Thank you!



Copyright

This document is distributed under the creative commons license 'Attribution-Noncommercial-No Derivative Works 3.0 Germany'

© 2010 flonatel GmbH & Co. KG

<http://www.flonatel.de/projekte/rmtoo>