Risk-Based Coverage Tool

Overview

The risk-based testing idea is to split test work into items, measure risk and test execution time associated with each item, and select resulting test coverage in such a way that the highest possible volume of risk is covered by test in a budget of time available to test org.

RBTCS tool (rbtcs.py) allows you to input risk-based coverage spreadsheet with risk assessment data and execution time estimates, and generates output excel file containing suggested optimized coverage for a given budget. The tool is capable of doing input validation, handling precondition relationship between items, and process seeding requirements specified by a user.

Please note that the process describing how to get risk-based coverage spreadsheet with risk assessment is outside of the scope of this project.

Please refer attachment for the latest version of the tool, and example files.

Prerequisites for using the tool

You need to have these items installed on your PC in order to use RBTCS tool:

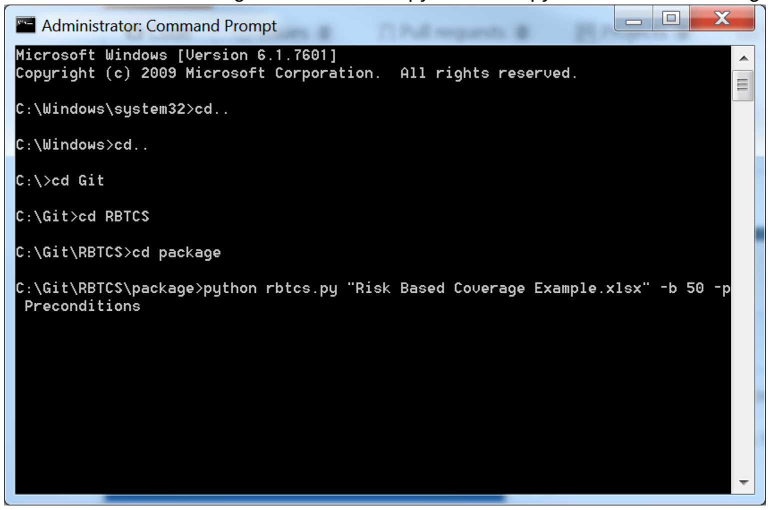
1. Python 2.7
2. Python libraries (use “pip install <libname>” to resolve):
    a. xlrd
    b. xlwt
    c. argparse
    d. logging
    e. enum

Main usage scenario

1. Test engineer prepares risk-based coverage spreadsheet (the latest version is attached to the test plan document template - <Release> Feature Test Plan <RQ#, Description>).
2. Test engineer runs the tool (rbtcs.py), and provides all necessary arguments in command line (the easiest way is to put rbtcs.py and spreadsheet file in the same folder).
    a. Option 1. If there are no precondition relations between functional requirements, the tool will use algorithms that guarantees optimal solution:
    `>python rbtcs.py "Risk Based Coverage Example.xlsx" -b 50`
    where Risk Based Coverage Example.xlsx is the input file name, and "50" is given budget of time.
    b. Option 2. If there are precondition relations between functional requirements and/or environmental requirements, the tool will use greedy approximation algorithm that produces good approximation of optimal solution:
    `>python rbtcs.py "Risk Based Coverage Example.xlsx" -b 50 -p Preconditions`
    where Risk Based Coverage Example.xlsx is the input file name, "50" is given budget of time, and "Preconditions" is a column name containing precondition lists for items.
3. Tool produces "rbtcs_results.xls" file which contains input file data but with "Covered (n)?" column filled (i.e. with data that shows which items have been selected for coverage - "y", and which items haven’t been selected - "n").
4. In the CLI output tool also shows risk coverage level for generated solution.
5. User should copy content of the "Covered (n)?" column from "rbtcs_result.py" file back to original file (for example, Risk Based Coverage Example Solution.xlsx contains coverage column copied from tool output).

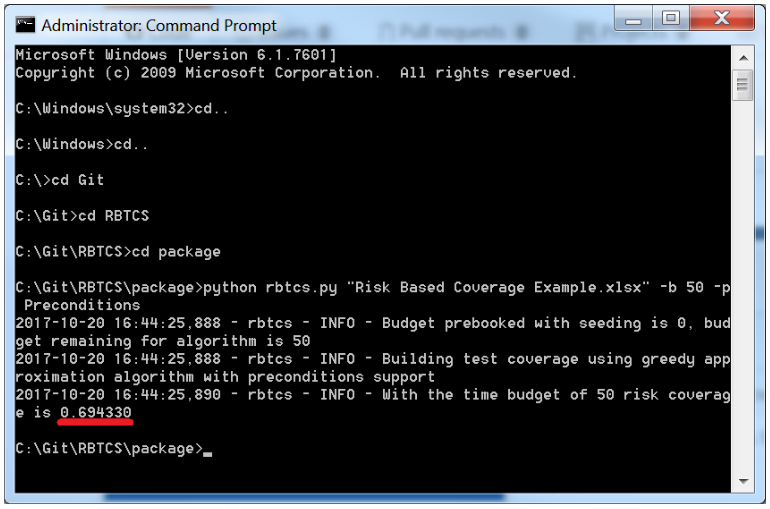
Usage example

1. Download and review "Risk Based Coverage Example.xlsx" file attached to this page.
2. Run the tool with following command line: >python rbtcs.py "Risk Based Coverage Example.xlsx" -b 50 -p Preconditions



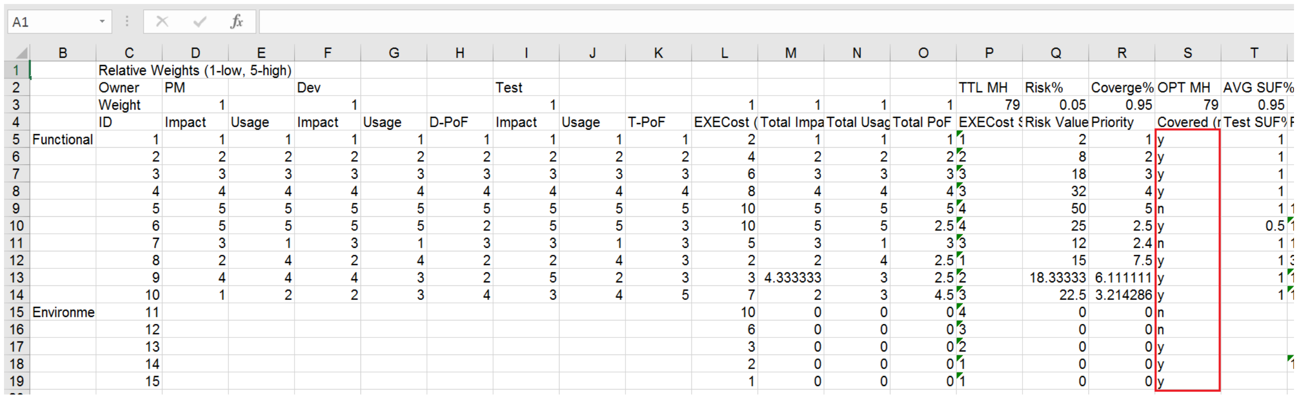
```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Windows\system32>cd..

C:\Windows>cd..

C:\>cd Git

C:\Git>cd RBTCS

C:\Git\RBTCS>cd package

C:\Git\RBTCS\package>python rbtcs.py "Risk Based Coverage Example.xlsx" -b 50 -p
 Preconditions
```

3. Observe command line output and output in "rbtcs_results.xls". (there is "Risk Based Coverage Example Solution.xlsx" attached to the page, where the tool output was copied back into example spreadsheet).



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Windows\system32>cd..

C:\Windows>cd..

C:\>cd Git

C:\Git>cd RBTCS

C:\Git\RBTCS>cd package

C:\Git\RBTCS\package>python rbtcs.py "Risk Based Coverage Example.xlsx" -b 50 -p
 Preconditions
2017-10-20 16:44:25,888 - rbtcs - INFO - Budget prebooked with seeding is 0, bud
get remaining for algorithm is 50
2017-10-20 16:44:25,888 - rbtcs - INFO - Building test coverage using greedy app
roximation algorithm with preconditions support
2017-10-20 16:44:25,890 - rbtcs - INFO - With the time budget of 50 risk coverag
e is 0.694330

C:\Git\RBTCS\package>_
```

Proposed test set covers 69.433% of total risk value.



A1

	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	
1		Relative Weights (1-low, 5-high)																		
2		Owner	PM		Dev			Test							TTL MH	Risk%	Coverge%	OPT MH	AVG SUF%	
3		Weight	1		1			1			1	1	1	1	79	0.05	0.95	79	0.95	
4		ID	Impact	Usage	Impact	Usage	D-PoF	Impact	Usage	T-PoF	EXECost (	Total Impa	Total Usag	Total PoF	EXECost S	Risk Value	Priority	Covered (r	Test SUF%	F
5	Functional	1	1	1	1	1	1	1	1	1	2	1	1	1	1	2	1	y	1	
6		2	2	2	2	2	2	2	2	2	4	2	2	2	2	8	2	y	1	
7		3	3	3	3	3	3	3	3	3	6	3	3	3	3	18	3	y	1	
8		4	4	4	4	4	4	4	4	4	8	4	4	4	3	32	4	y	1	
9		5	5	5	5	5	5	5	5	5	10	5	5	5	4	50	5	n	1	1
10		6	5	5	5	5	2	5	5	3	10	5	5	2.5	4	25	2.5	y	0.5	1
11		7	3	1	3	1	3	3	1	3	5	3	1	3	3	12	2.4	n	1	1
12		8	2	4	2	4	2	2	4	3	2	2	4	2.5	1	15	7.5	y	1	3
13		9	4	4	4	3	2	5	2	3	3	4.333333	3	2.5	2	18.33333	6.111111	y	1	1
14		10	1	2	2	3	4	3	4	5	7	2	3	4.5	3	22.5	3.214286	y	1	1
15	Environme	11									10	0	0	0	4	0	0	n		
16		12									6	0	0	0	3	0	0	n		
17		13									3	0	0	0	2	0	0	y		
18		14									2	0	0	0	1	0	0	y		1
19		15									1	0	0	0	1	0	0	y		

"Covered (n)?" column in "rbtcs_result.xls" contains proposed coverage.
Please note that tool output file (rbtcs_result.xls) doesn't carry formatting and formulas from original spreadsheet. It it suggested just to copy "Covered (n)?" column back to original file, and proceed with coverage review using original file only.

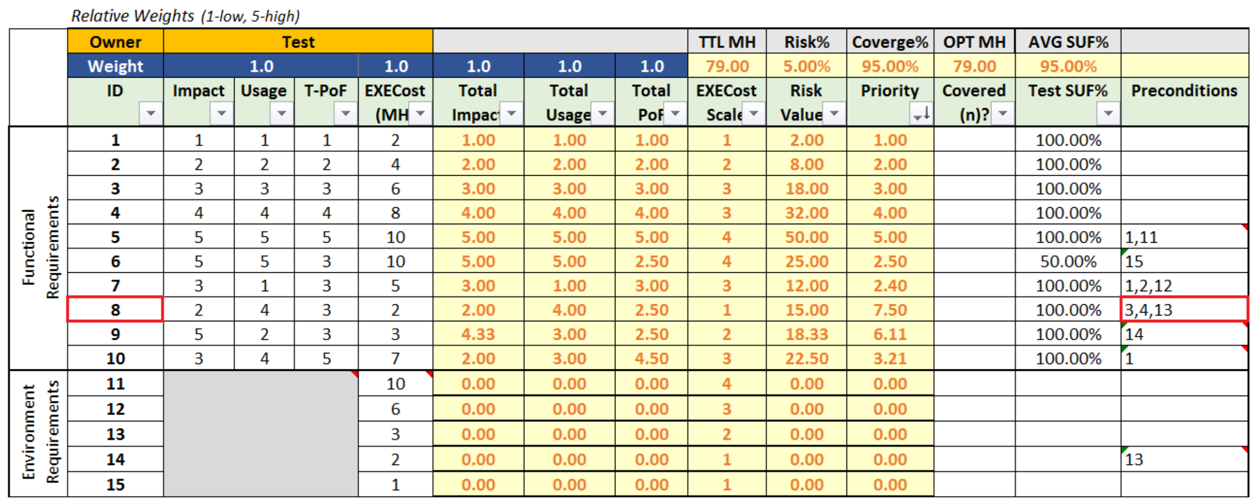
Input data file specification

Tool works with "Risk Based Coverage.xlsx" spreadsheet format. And it has certain data requirements:

1. The data should be on the very first sheet of the document.
2. Do not alter column names, use names suggested in the spreadsheet template document.
3. The tool interprets rows following the header row as row containing items (i.e. functional requirements or environmental requirements).
4. "Risk Value" column content should be either integer or float (for example, "2" or "2.00").
5. "EXECost (MH)" column content should be integer (for example, "10").
6. "Covered (n)?" column can be used to provide seeding data. Use 'y' symbol to pre-select item (positively seed an item), or 'n' symbol to pre-exclude item (negatovely seed an item).
7. "Preconditions" column should contain comma-separated lists of preconditions, each precondition in a list should be an integer value (for example, "1,2,3" meaning that items #1, 2, and 3 are precondition of particular item). Precondition value range between 1 and N, where N is the total number of items in an input file.

Preconditions

Preconditions are a form of relationship between items. Whenever one choses an item with preconditions, those precondition items have to be added to final coverage as well. To specify preconditition for item A, you should provide a list of comma-separated integer values in "Preconditions" column for item A of the spreadsheet.



Relative Weights (1-low, 5-high)

	Owner	Test							TTL MH	Risk%	Coverge%	OPT MH	AVG SUF%	
	Weight	1.0			1.0	1.0	1.0	1.0	79.00	5.00%	95.00%	79.00	95.00%	
	ID	Impact	Usage	T-PoF	EXECost (MH	Total Impact	Total Usage	Total PoF	EXECost Scale	Risk Value	Priority	Covered (n)?	Test SUF%	Preconditions
Functional Requirements	1	1	1	1	2	1.00	1.00	1.00	1	2.00	1.00		100.00%	
	2	2	2	2	4	2.00	2.00	2.00	2	8.00	2.00		100.00%	
	3	3	3	3	6	3.00	3.00	3.00	3	18.00	3.00		100.00%	
	4	4	4	4	8	4.00	4.00	4.00	3	32.00	4.00		100.00%	
	5	5	5	5	10	5.00	5.00	5.00	4	50.00	5.00		100.00%	1,11
	6	5	5	3	10	5.00	5.00	2.50	4	25.00	2.50		50.00%	15
	7	3	1	3	5	3.00	1.00	3.00	3	12.00	2.40		100.00%	1,2,12
	8	2	4	3	2	2.00	4.00	2.50	1	15.00	7.50		100.00%	3,4,13
	9	5	2	3	3	4.33	3.00	2.50	2	18.33	6.11		100.00%	14
	10	3	4	5	7	2.00	3.00	4.50	3	22.50	3.21		100.00%	1
Environment Requirements	11				10	0.00	0.00	0.00	4	0.00	0.00			
	12				6	0.00	0.00	0.00	3	0.00	0.00			
	13				3	0.00	0.00	0.00	2	0.00	0.00			
	14				2	0.00	0.00	0.00	1	0.00	0.00			13
	15				1	0.00	0.00	0.00	1	0.00	0.00			

For example, item #8 has items #3, #4, and #13 as preconditions. Items #3-4 are functinal preconditions, and item #13 is environmental precondition. This means that in order to cover functionality associated with functional requirement 8, functional requirements 3 and 4 have to be in coverage too, and you need environment specified in item 13 as well.

The tool assumes that all items are number contiguously starting from 1. Whenever you list a number in "Preconditions" column, the tool assumes that you refer item matching this number as a precondition. The tool will fail if you use discontiguous item numbering.

There is input validation for values in "Preconditions" column:

1. It should be comma-separated list of integers;
2. Integer values should be greater than or equal to 1;
3. Integer values should be less than or equal to N, where N is the overall number of items in the input file.
4. It is OK to have preconditions specified out of order (for example, "3,100,17").
5. The tool can handle duplicates in precondition lists (for example, "1,3,5,3,10").

Seeding

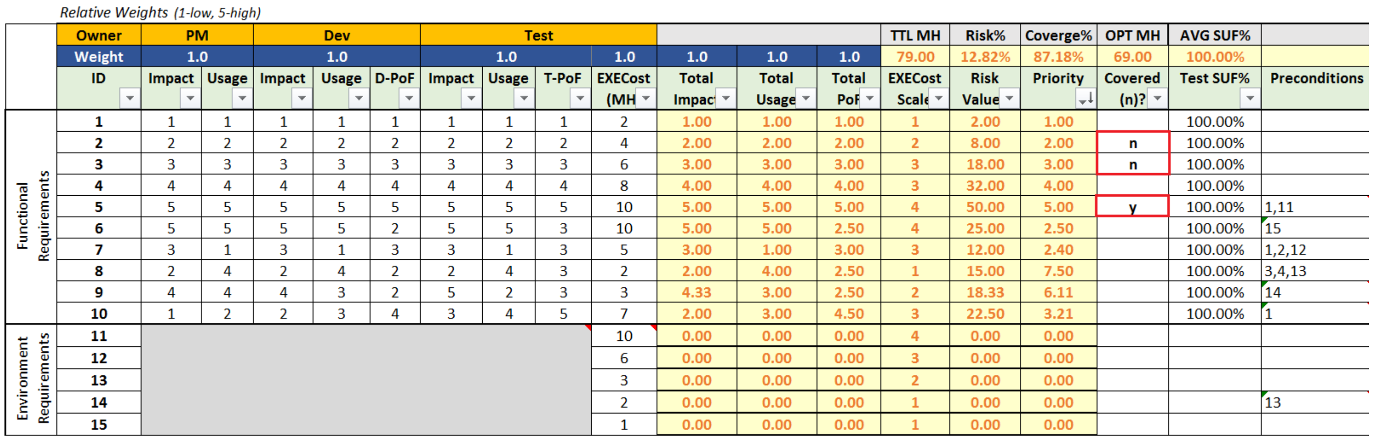
Seeding feature allows you to pre-select or pre-exclude certain items from final coverage. Implemented algorithms will honor seeding.

- Pre-selection of certain items is a positive seeding. Positively seeded item MUST be included in final coverage.
- Pre-exclusion of certain items is a negative seeding. Negatively seeded items MUST NOT be included in final coverage.

One can use "Covered (n)?" column to specify seeding data (see example below):

- If you want to include an item into a final coverage (positive seeding) - use **'y'** symbol in corresponding cell of "Covered (n)?" column.

- If you want to exclude an item from a final coverage (negative seeding) - use **'n'** symbol in corresponding cell of "Covered (n)?" column.



Relative Weights (1-low, 5-high)

	Owner	PM		Dev			Test							TTL MH	Risk%	Coverge%	OPT MH	AVG SUF%	
	Weight	1.0		1.0			1.0			1.0	1.0	1.0	1.0	79.00	12.82%	87.18%	69.00	100.00%	
	ID	Impact	Usage	Impact	Usage	D-PoF	Impact	Usage	T-PoF	EXECost (MH	Total Impact	Total Usage	Total PoF	EXECost Scale	Risk Value	Priority	Covered (n)?	Test SUF%	Preconditions
Functional Requirements	1	1	1	1	1	1	1	1	1	2	1.00	1.00	1.00	1	2.00	1.00		100.00%	
	2	2	2	2	2	2	2	2	2	4	2.00	2.00	2.00	2	8.00	2.00	n	100.00%	
	3	3	3	3	3	3	3	3	3	6	3.00	3.00	3.00	3	18.00	3.00	n	100.00%	
	4	4	4	4	4	4	4	4	4	8	4.00	4.00	4.00	3	32.00	4.00		100.00%	
	5	5	5	5	5	5	5	5	5	10	5.00	5.00	5.00	4	50.00	5.00	y	100.00%	1,11
	6	5	5	5	5	2	5	5	3	10	5.00	5.00	2.50	4	25.00	2.50		100.00%	15
	7	3	1	3	1	3	3	1	3	5	3.00	1.00	3.00	3	12.00	2.40		100.00%	1,2,12
	8	2	4	2	4	2	2	4	3	2	2.00	4.00	2.50	1	15.00	7.50		100.00%	3,4,13
	9	4	4	4	3	2	5	2	3	3	4.33	3.00	2.50	2	18.33	6.11		100.00%	14
	10	1	2	2	3	4	3	4	5	7	2.00	3.00	4.50	3	22.50	3.21		100.00%	1
Environment Requirements	11									10	0.00	0.00	0.00	4	0.00	0.00			
	12									6	0.00	0.00	0.00	3	0.00	0.00			
	13									3	0.00	0.00	0.00	2	0.00	0.00			
	14									2	0.00	0.00	0.00	1	0.00	0.00			13
	15									1	0.00	0.00	0.00	1	0.00	0.00			

In this example, items #2-3 were negatively seeded, and item 5 was positively seeded.

Using seeding items, be aware that seeding contradictions and seeding oversubscription may happen.

- Seeding contradiction is a situation when you positively seed item A, which has item B as a precondition. And you negatively seed item B. In this case seeding options for item A and B contradicts each other. The tool will detect such contradictions and log appropriate message specifying details (i.e. seeding for which items raised contradiction).
- Seeding oversubscription is a situation when total budget required for all positively seeded item (both explicitly and implicitely seeded) exceeds budget specified by "-b" command line option.

Finally, there can be explicit and implicit seeding:

- Explicit positive seeding are items that you specified with 'y' symbol in your input file. Item #5 on the picture above was explicitly positively seeded.
- Implicit positive seeding are items that the tool has to include into final coverage, because they were preconditions for explicitly positively seeded items. Items #1 and #11 will be implicitly positively seeded items for example above, because both of them were listed as precondition for explicitly positively seeded item #5.
- Explicit negative seeding are items that you specified with 'n' symbol in your input file. Item #2-3 on the picture above were explicitly negatively seeded.
- Implicit negative seeding are items that the tool has to exclude from final coverage, beacuse items you negatively seeded were preconditions for them. Items #7 and #8 will be implicitly negatively seeded items for example above, because item #7 has item #2 listed as precondition, and item #8 has item #3 listed as precondition.
- Please note that the tool will track all chains of precondition relationship to properly detect all implicit seeding or contradictions. For example, if you explicitly positively seeded item A, and item A has item B as precondition, and item B has item C as precondition, then both B and C will be implicitly positively seeded items. Similar logic applies for negative seeding and contradiction checks.

Algorithms that calulate coverage

The tool currently supports three algorithms and choose one automatically.

- The first algorithm is based on dynamic programming solution for 01 knapsack problem and it guarantees optimal solution. This algorithm is used in case if NO preconditions specified for the input data (i.e. no "-p" argument provided in CLI). In case if number of items and specified time budget are high (for example, 300 items with total budget of 15000), it is likely that Memory Exception will happen during algorithm calculations. This event will be handled by the tool, and simple version of greedy algorithm will be triggered.
- Simple greedy algorithm is used when there are no preconditions specified (i.e. no "-p" argument provided in CLI), and memory exception happened during dynamic programming algorithm computation. The greedy algorithm will compute relative risk to cost ratio for each item (ratio[i] = risk[i]/cost[i]), sort all items in descending order of relative ratio, and put as many items going from the top to the bottom of sorted item list as possible.
- The third algorithm is a greedy algorithm with support of precondition relationship between items. It is used when preconditions are specified (i.e. you specified precondition column name in "-p" CLI argument). It's idea is close to idea of simple greedy algorithm, however whenever it calculates relative ratio, it checks and uses all preconditions of a particular item. When the item is selected for inclusion, all it's preconditions (that haven't been yet added into final coverage) are also included. And the algorithm recomputes relative ratio for remaining items and make next walk for the next candidate.

Logging messages

Current version has logging level set to INFO (there are 6 level total: CRITICAL, ERROR, WARNING, INFO, DEBUG, NOTSET).

All log messages have the following format:

```
<date> <time> - rbtcs - <log level> - <log message>
```

For example:

```
2017-09-06 11:08:06,812 - rbtcs - CRITICAL - illegal input file name or file doesn't exist
```

Logging messages:

LEVEL	**Message**	**Conditions**
CRITICAL	“%s is illegal input file name or file doesn't exist”	When %s argument specified in command line as a source file name doesn’t exist or represent illegal name.
CRITICAL	"Header row not found!"	Message generated when header row wasn’t found in an input file. Column names used by the tool (default values or values that you specified using -r, -t, -s, -p CLI keys) doesn't match input file.
CRITICAL	"Header row can't be the last row in the file!"	Message generate when header row is the last row on an input sheet. There should be item rows after the header row.
DEBUG	“Header row index: %d”	Message generated when header row detected. %d specifies the input file row number corresponding to header row.
DEBUG	“Risk Factor column index: %d”	Message generated when header row detected. %d specifies the input file column number corresponding to risk factor column.
DEBUG	“Execution Time column index: %d”	Message generated when header row detected. %d specifies the input file column number corresponding to execution time column.
DEBUG	"Selection column index: %d"	Message generated when header row detected. %d specifies the input file column number corresponding to selection column.
DEBUG	"Preconditions column index: %d"	Message generated when header row detected. %d specifies the input file column number corresponding to preconditions column.
CRITICAL	“Time budget is not a positive number: %d”	Message is generated when a value of time budget specified in --time-budget TIME_BUDGET argument is not a positive number. %d specifies budget provided.
CRITICAL	“Can't convert Risk Factor for item in row # %d to float”	Message appears when the tool is unable to convert value of risk factor for test cases with index %d into float. %d here specifies the item row number in input file.
CRITICAL	“Can't convert Execution Time for item # %d to integer”	Message appears when the tool is unable to convert value of execution time for test cases with index %d into integer. %d here specifies the item row number in input file.
CRITICAL	"Can't convert Preconditions string for item # %d to list of integers "	Message appears when the tool is unable to convert specified preconditions for test cases with index %d into a list of integers. %d here specifies the item row number in input file.

CRITICAL	"Precondition value \'%d\' for item # %d is too big or too small"	Message appears when one of specified preconditions is smaller than 1 or greater than N (where N is the total number of items in your input file). In other words, precondition should refer one of existing items.
WARNING	“Specified time budget is relatively big which may prevent from getting optimal solution”	Message is generated when specified time budget exceeds internal threshold MAX_BUDGET (currently it is equal to 10000). Such a big budget may prevent from getting optimal solution due to potentially high space complexity.
WARNING	"Number of Items in the input file is relatively big which may lead to sub-optimal solution"	Message is generated when number of items in your input file exceed internal threshold MAX_ITEMS (currently, it is 300). Such a big input may prevent from getting optimal solution.
CRITICAL	"Error reading input file in XLRD"	Message is generated when there was an error in XLRD library while reading input file. (note: XLRD library is used to read input .xls and .xlsx files).
DEBUG	“XLRD Exception: <exception string>"	Message is generated when there was an error in XLRD library while reading input file. And it contains exception string returned by XLRD library for debug purposes.
INFO	“Building test coverage using optimal algorithm”	Message appears when the tool tries to calculate optimal solution for input data (which has been read and validated successfully). Optimal algorithm is used only when there are no precondition relationships between items.
INFO	"With the time budget of %d risk coverage is %f"	Message appears if an algorithm succeeded. %f value is a value of risk covered achieved with selected items using given budget of %d.
ERROR	“Caught MemoryError exception while building test set using dynamic programming algorithm for 01 knapsack problem”	Message appears when space complexity of optimal algorithm is it too big and led to memory exception.
INFO	“Building test coverage using greedy approximation algorithm”	Message specifies that greedy algorithm was triggered after optimal algorithm generated memory exception. Still in case if there are no precondition relationships between items.
INFO	"Building test coverage using greedy approximation algorithm with prerequisites support"	Message appears you instructed the tool that there are preconditions in your input file. Greedy algorithms with precondtion support is triggered.
CRITICAL	"Error writing results file in XLWT"	Message is generated when there was an error in XLWT library while writing output file. (note: XLWT library is used to write output into .xls).
DEBUG	"XLWT Exception: <exception string>"	Message is generated when there was an error in XLWT library while writing output file. And it contains exception string returned by XLWT library for debug purposes.
CRITICAL	"Contradiction detected for seeding data: item #%d was selected by user, and it has precondition item #%d which was excluded by user"	Message is generated when a precondition contradiction is detected. The numbers of items that led to contradiction are provided. Please note that specified items can either be explicitly seeded or implicitly seeded).

CRITICAL	"Contradiction detected for seeding data: budget of all items selected by user is %d and it exceeds available time budget of %d",	Message is generated when budget oversubscription by positively seeded items is detected. Please note that both explicitly and implicitly seeded items are taken into consideration.
INFO	"Budget prebooked with seeding is %d, budget remaining for algorithm is %d"	Shows prebooked and remaining budgets.
INFO	"%d items were identified as explicitly excluded by user"	Logs number of explicitly negatively seeded items.
INFO	"Additionally, %d items were identified as implicitly excluded by user"	Logs number of implicitly negatively seeded items.
INFO	"%d items were identified as explicitly selected by user"	Logs number of explicitly positively seeded items.
INFO	"Additionally, %d items were identified as implicitly selected by user"	Logs number of implicitly positively seeded items.

Command line arguments

Command line format:

```
python rbtcs.py [-h] [-r RISK_FACTOR] [-t EXECUTION_TIME] [-s SELECTION] [-b TIME_BUDGET] [-p PRECONDITIONS] filename
```

Mandatory arguments:

- filename – specifies name of input file (either absolute path and name, or relative)

Optional arguments:

- -h – show help
- -r RISK_FACTOR – specifies name of risk-factor column as RISK_FACTOR value. Default value is “Risk Values”.
- -t EXECUTION_TIME – specifies name of execution-time column as EXECUTION_TIME value. Default value is “EXECost (MH)”.
- -s SELECTION – specifies name of selection column as SELECTION value. Default value is “Covered (n)?”.
- -b TIME_BUDGET – specifies overall time budget as TIME_BUDGET value. Default value is 2500.
- -p PRECONDITIONS – specifies preconditions column name. Default value is empty, i.e. no preconditions usage by default. Please note that risk-based coverage worksheet uses "Preconditions" column name for precondition relationship specification.