R Package for Reliability Research

### Github to Jace’s Code

<https://github.com/WarrRich/Assurance-Testing>

### Writing an R Package

<https://ourcodingclub.github.io/tutorials/writing-r-package/>

### Deliverables

**Deliverables:**

Cleanly formatted Rpackage code

Read.md file associated with code

Descriptions associated with each function

Vignette page for code walkthrough

**Main package/function input:**

The user's system as an RBD

* Can either be done within the user’s code chunk, or allow them to read in a separate file…

Series, parallel, phantom vs no phantom (this should be defined in the step above)

Given the function FindRisksSys, output the number of iterations, so the user knows where in the algorithm they are (more internal--users don't need to worry about this)

Component costs (written in code as prices), system cost (sys\_only\_price) (should be incorporated in the RBD in step 1)

Rbeta parameters (alpha, beta) (should be incorporated in the RBD in step 1)

Rejectable reliability level, acceptable reliability level

Number of monte carlo samples (i.e. 100000) (mcsamps) (we might want to set a default the users can change)

A “method=” should be provided to try out this other algorithm

**Main package/function output:**

Number of acceptable system failures (cvec)

Number of assurance tests applied (nvec)

And test costs

PPR and PCR (more internal--unless the user asks)

Comparison between System Tests Only, System & Component Tests, Cost Difference, and Time Difference between the two (only if the user asks)

Output for each iteration (only if the user asks)

**Internal workings the users don't need to see:**

Test Score (to establish a preference for one test over another)

### Working Vignette

1 Introduction

The goal of the package is to reduce the overall cost of an assurance test by augmenting system tests with component tests. We take a straight-forward approach by first finding an assurance test that is composed solely of system tests. This is the baseline from which we start; and the goal is to improve (i.e., lower) the cost of this baseline test plan. In this vignette we walk through the basics of a simple system assurance test and how to utilize the functions correctly.

2 Our system

The function find\_risks\_system() is meant to output three things: the number of acceptable system failures (cvec), the number of assurance tests applied (nvec) , and test costs. It will allow the user to make comparisons between the system tests only, system & component tests, and cost difference. None of this function would be possible without providing system information via a RBD (reliability block diagram). For this example, we'll proceed with a simple bike example:

In this case, our bike will be made up of two subsystems: tires and brakes.

S(BackTire, FrontTire):Tires

P(FrontBrake, BackBrake): Brakes

S(Tires, Brakes): Bike

In this case, BackTire and FrontTire are series components part of the subsystem Tires, Front Brake and BackBrake are parallel components part of the subssytem Brakes, and Tires and Brakes are series subsystems part of the system Bike.

In order to provide all of the necessary information for the function find\_risks\_system(), the syntax should be as follows:

*"SYSTEM NAME: subsystem relationship (either parallel or series), $cost of system test*

*SubsystemName1: $cost of subsystem test*

*SubsystemName2: $cost of subsystem test*

*SubsystemName1: component relationship (either parallel or series)*

*Component1.1: beta priors, $cost of component test*

*Component1.2: beta priors, $cost of component test*

*SubsystemName2: component relationship (either parallel or series)*

*Component2.1: beta priors, $cost of component test*

*Component2.2: beta priors, $cost of component test"*

*P* or *S*( \**SUBSYSTEM NAME\*:$cost of system test:,*

*\*Component Name\*:*prior=*B(a,b)*:testable=*TRUE* or *FALSE:$cost of component test:)*

Component and subcomponent words/names are distinguished between asterisks (\*). The cost of system or subsystem tests are preceded by a dollar sign $. Beta prior values a and b from which we will sample from are written in the form B(a,b). In the case of an untestable lowest level component, the cost would be denoted $Inf.

Using our Bike example with fictitious numbers:

file <- textConnection("Bike:: series: cost=$45

Tires: cost=$33

Brakes: cost=$45

Tires:: series

BackTire: B(1,3), cost=$12

FrontTire: B(1,2), cost=$12

Brakes:: parallel

FrontBrakes: B(3,2), cost=$12

BackBrakes: B(4,2), cost=$Inf")

The capitalization of the subsystems and system name, the indentation, and the spacing are for readability purposes, and can be included per the user's choice. System or component names may contain numbers or letters.

Say that for the Bike system, it also requires the components Bell and KickStand to be added in a series relationship, but not belonging to a subsystem. Notation for such a system would look like:

file <- textConnection("Bike:: series: cost=$245

Tires: cost=$33

Brakes: cost=$45

Bell: B(1,3), cost=$19

KickStand: B(2,3), cost=$33

Tires:: series

BackTire: B(1,3), cost=$12

FrontTire: B(1,2), cost=$12

Brakes:: parallel

FrontBrakes: B(3,2), cost=$12

BackBrakes: B(4,2), cost=$Inf")

Brakes:: parallel

FrontBrakes: B(3,2), cost=$12

BackBrakes: B(4,2), cost=$Inf")

### Consider Replacing the Bicycle Example?

### I LOVE the bike example. What are some other examples we could use?

## CPU and RAM (series)

## Online website example:

S(Frontend, Backend): Website

P(Inventory, Payment): Backend

S(GUI, UserAuthentication): Frontend

### Home security system

S(Sensors, Alarms): Security System

P(LoudSiren, Silent): Alarm (loud siren scares off intruders, silent notifies authorities)

S(Frontdoor, Backdoor): Sensors

### Dual drive Electric scooter

S(FrontMotor, RearMotor): Propulsion

P(FrontBrake, BackBrakes): Brakes

S(Propulsion, Brakes): Scooter

### Working Function Description

|  |  |
| --- | --- |
| component\_augmented\_testing {CompAugTesting} | R Documentation |

Creates a component-augmented cost-effective plan for a given system's reliability testing.

**Description**

The user provides a reliability block diagram of the system that includes testing costs and component prior information. Given the posterior producer’s risk, the posterior consumer’s risk, the rejectable reliability level, and the acceptable reliability level, the software returns a cost-effective test plan. The test plan is defined by the number of assurance tests needed and the number of acceptable failures for each component, subsystem, and the system. It also provides the cost savings compared with an assurance test comprised solely of system tests.

**Usage**

component\_augmented\_testing(file\_or\_file\_path, alpha, beta, cvec, nvec, p.rrl, p.arl, mcsamps, samp)

**Arguments**

|  |  |
| --- | --- |
| file | A string or the file path containing the reliability diagram (see details) |
| alpha | α maximum acceptability probability threshold for the producer's risk |
| beta | β the maximum acceptable probability thresholds for the consumer’s risk |
| p.rrl | A specified rejectable reliability level |
| p.arl | A specified acceptable reliability level |
| mcsamps | Number of Monte Carlo samples generated from beta random distribution |
|  |  |

**Value**

For the output, user can expect a cost-effective test plan to be returned, where the number of tests for the system, components, and acceptable failures with total cost.

**file Details**

The file is a text file specifying your reliability block diagram, and should also include the costs of the system and component tests, as well as the priors of the component tests. Here is the structure and required syntax:

"

SYSTEMNAME::SeriesOrParallel:cost=$\_\_\_\_

SUBSYSTEMNAME1: cost=$\_\_\_\_\_

SUBSYSTEMNAME2: cost=$\_\_\_\_

\n

SUBSYSTEMNAME1::SeriesOrParallel

COMPONENT1.1: B(prior\_alpha,prior\_beta), cost=$\_\_\_\_

COMPONENT2.1: B(prior\_alpha,prior\_beta), cost=$\_\_\_

\n

SUBSYSTEMNAME2::SeriesOrParallel

COMPONENT2.1: B(prior\_alpha,prior\_beta), cost=$\_\_\_

COMPONENT2.2: B(prior\_alpha,prior\_beta), cost=$\_\_\_"

If the test cost is included we assume the subsystem is testable. If cost=$Inf, the function assumes it is not testable, or the cost is unknown.

Parallel and series subsystems can be specified as in the following example:

"

BI2KE::series:cost=$1999

Tires: cost=$33

Brakes: cost=$45

Tires::Series

BackTire: B(1,2), cost=$12

FrontTire: B(1,2), cost=$12

Brakes::series

FrontBrakes: B(3,2), cost=$12

BackBrakes: B(1,4), cost=$Inf"

The first word in each grouping is the system or subsystem name. The first section (where sections are denoted by new lines) represents the system information. The system "Bike" is made up of two subsystems: "Tires" and "Brakes". Tires and Brakes are related by a series relationship, meaning that both need to work in order for the system to function.

S and P represent series and parallel relationships respectively. For example, the last section of this diagram can be read as BackBrakes and FrontBrakes are components in parallel of a subsystem Brakes.

The following syntax conventions for the file will be enforced:

* System information must be presented in the first section (where sections are recognized by \n or new line).
* The system and subsystems also must be followed by a double colon when describing their components’ information.
* Each system or subsystem must contain whether it’s in a series or parallel relationship.
* Components can be uppercase, lowercase, and include numbers, but no special characters (names are case-sensitive).
* Two or more elements can be specified in a relationship.
* The priors a and b will be recognized by B(a,b) for each lowest level component.
* The costs of each test will be recognized by cost=$\_\_\_, and untestable components will be recognized by cost=$Inf.
* Currently, all component names must be unique.

**Examples**

<!--you’ll put in Run examples -->

### Next steps:

* Remove TIP\_function from find\_risk\_sys
  + Create TIP to rerun to work in flowchart from figure 6 in paper
* Create figure 6 to find total system test
* Implement Algorithm 1