Random DDP Model Instance Generator

Andres Orrego
Global Science & Technology, Inc.
Fairmont, WV, USA
andres.orrego@gst.com

ABSTRACT

Currently, the number of DDP models generated at JPL is very limited due to the relatively short period of time these models have been around and the long time it takes to get the right project people to meet and discuss requirements, risks, and mitigations. In order to find the best search method to find the best solution to these models, more instances need to be explored and therefore an instance generator is required. Such instance generator has to follow the constraints set by the model and randomize the values of its variables so the external validity of studies done using them is not compromised. This paper explains the rules followed for the implementation a DDP model instance generator.

Keywords: Random generator, DDP models, AI search

1. INTRODUCTION

In recent years WVU and JPL have partnered together to study ways to improve software development processes. One example of such effort is the development and research of DDP models [1]. A DDP model defines the relationship between requirements, risks, and risk mitigation strategies so they can be evaluated intelligently to reduce project costs while achieving the maximum requirements coverage. Generating these models requires a series of long meetings among the best project engineers at JPL to collect requirements, identify risks and discuss the cost and impact of mitigation strategies so the most requirements are achieved at the lowest cost. Finding the particular point where this goal is maximized is not a trivial task as the solution space grows exponentially based on the number of requirements, risks, and mitigations. It would be impossible for humans to search such vast space and therefore machines need to be used to test for the best solution. In some instances, when the models are not very big, it is possible to explore all possibilities and find the best solution that satisfies our goal but some other times this task becomes time-prohibitive and heuristic search based software engineering methods are required.

2. BACKGROUND

The defect detection and prevention (DDP) approach was first invented in 1998 by Steven Cornford, at the Jet Propulsion Laboratory. It is a risk-based requirements model that assists in early lifecycle decision making to help developers select assurance activities in a cost-effective maner. That is, maximizing requirements attainment while minimizing mitigation costs. This model, as we study it, is based on three concepts: requirements, risks, and mitigations. Values are assigned to each of these factors to reflect importance, likelihood, and cost respectively. Each requirement is assigned a numeric *Weight* ranging from 0 to a MAXWEIGHT, usually 100. This number denotes the priority of the requirement in terms of how important it is to attain it compared to other requirements. In terms of risks, each one of them is assigned a likelihood indicating the probability of its occurrance in case no mitigation is exercised. This a-priori likelihood or *rAPL* is measured as a floating-point number ranging from 0 to 1 . Lastly, each mitigation is assigned a *Cost* which is usually the financial cost it would take to take the steps necessary to prevent a risk (or risks) from happening. Mitigations are also assigned a boolean, *Selected*, that is set to *true* it will be performed, *false* otherwise. In additon to these factors, DDP models also consider the relationships among them. For instance, risks and requirements are related in that if the former occurs, the attainment of the latter is negatively impacted. Given that our goal is to maximize requirement, this impact is measured as the loss of attainment imposed by the risk should it occur in floating-point values ranging from 0 to 1 (inclusive). An *Impact(risk,requirement)* value of 0.5 means that should the risk happen, the requirements attainment is reduced by one half. A second relationship is established between risks and mitigatins. The *effect* of a mitigation indicates how much it reduces each risk, and it is also measured in decimal values ranging from 0 to 1 inclusive. An *Effect(mitigation,risk)* value of 0.5 means that the given mitigation reduces the risk by one half. Given these factors and the relationships among them we can then search for the best way to control them to achieve the maximum requirements attainment at the lowest mitigation costs. Keep in mind that maximizing attainment requires minimizing risks by implementing costly mitigations. At the same time, minimizing costs prevents projects from implementing mitigations to reduce risks.

3. INSTANCE GENERATOR

We developed our instance generator using (almost) pure bash shell scripting, simplifying the source code but limiting performance and constraining the values ranges for certain model parameters. In this section we summarize the factors that compose DDP models and the values they can take in our random generator.

We start by seting the three main factors that set the size of the model: requirements (or objectives), risks, and mitigations. The number of requirements, risks and mitigations vary from project to project. Small subsystems may have only few number of these fac-

tors, while a complete spacecraft software project may have hundreds. For the purposes of generating small to medium size random models we establish the number of requirements *(O_COUNT)*, risks *(R_COUNT)*, and mitigations *(M_COUNT)* to be integers ranging from 1 to 200. We allow the user to specify these numbers from the command line.

$$\begin{aligned} O\_COUNT &: USER\_DEFINED[1..200] \\ R\_COUNT &: USER\_DEFINED[1..200] \\ M\_COUNT &: USER\_DEFINED[1..200] \end{aligned}$$

The next step in the construction of the "model.h" file, which contains the model, is the definition of the DDP data structure. Since this step is immutable, that is the DDP data structure does not change, so all we do is print the c code for the structure, allocate memory to hold the structure, and begin to form the model setup "SetupModel()" function with an array of mitigation costs "*ddpData->mCost*".

For each mitigation we randomly assign a cost ranging from 0 to 32767 (signed 16-bit integer). We do this by creating an array of size MCOUNT, looping through it sequentially and assigning a random value from the $RANDOM variable.

$$\begin{aligned} mCost[M_1] &= RANDOM[0..32767] \\ mCost[M_2] &= RANDOM[0..32767] \\ \vdots &\quad \vdots \\ mCost[M_{M\_COUNT}] &= RANDOM[0..32767] \end{aligned}$$

The limited range for cost is the first constraint imposed by bash but it has a limited impact on the model as this variable can be measured in dolars, thousands of dollars, millions of dollars, etc.

We assign an a-priori likelihood (*ddpData->rAPL[x]*) of 1 to each one of the identified risks as another initialization step in "SetupModel()." Again, a one-dimensional array is used for this purpose.

$$\begin{aligned} rAPL[R_1] &= 1 \\ rAPL[R_2] &= 1 \\ \vdots &\quad \vdots \\ rAPL[R_{R\_COUNT}] &= 1 \end{aligned}$$

For each one of the risks we assign weights (oWeight) ranging from 0 to 100 (inclusive) at random.

$$\begin{aligned} oWeight[O_1] &= RANDOM[0..100] \\ oWeight[O_2] &= RANDOM[0..100] \\ \vdots &\quad \vdots \\ oWeight[O_{O\_COUNT}] &= RANDOM[0..100] \end{aligned}$$

In terms of the relationships between the factors, not every factor in the relationship is related to every other factor. For instance, not all risks have an impact on all requirements, therefore the risks and their impact-related requirements (roImpact[R][O]) are chosen at random with a 50% chance that a risk impacts a requirement. For all the selected roImpact relationships a decimal value from 0 to 1 is randomly assigned. Here it is the pseudo-code for this selection:

```
for i in 1..R_COUNT
{
  for j in 1..O_COUNT
  {
    hasimpact = RANDOM[true,false]
    if hasImpact is true
    {
      roImpact[i][j] = RANDOM[0,1]
    }
  }
}
```

A similar approach is used for the effect mitigations have on risks. Not all mitigations can reduce all risks and therefore our generator randomly selects the factors involved in this relationship at a 50% probability. Furthermore, the mrEffect relationship typically takes values in the range [0,1] but sometimes the effect of a mitigation can be detrimental to a risk. in other words, trying to prevent a risk from happening might increase the chance of another risk. In those special cases the mrEffect can be negative. Even in more special circumstances, the mrEffect can take a value lesser than -1. In these cases the mitigation involved not only increases the likelihood of the given risk, but it actually increases the impact that risk has on requirements. Given the special circumstances where mrEffect takes such values, our generator selects a decimal value from 0 to 1 at random and at 5% probability it multiplies it by -1, effectively making it a negatve number from 0 to 1. As a third step, at a 25% probability it substracts 1 from the negative value making it an aggravated risk (rAggrevated). The following is the pseudo-code for this step.

```
for i in 1..M_COUNT
{
  for j in 1..R_COUNT
  {
    hasImpact = RANDOM[true,false,0.5]
    if hasImpact
    {
      mrEffect[i][j] = RANDOM[0,1]
      isNegative = RANDOM[true,false,0.05]
      if isNegative
      {
       mrEffect[i][j] = mrEffect[i][j] * (-1)
       isGrave = RANDOM[true,false,0.25]
       if isGrave
       {
         mrEffect[i][j] = mrEffect[i][j] - 1
        }
      }
    }
  }
}
```

As explained above, at this point is when the model becomes complex. We start generating the array of risk likelihoods where, depending on the value mrEffect[M_i][R_j] takes, it may affect the requirements attainment (*oAttainment[O_k]*) by creating a risk aggravated impact (*rAggrevatedImpact[R_i]*) or by reducing the risk likelihood as follows:

- **If** $mrEffect[M_i][R_j] < -1$ **then** $rAggrevatedImpact[R_j]$ is increased by multiplying its current value by $(1 - m[M_i] * 1 + mrEffect[M_i][R_j])$ for each mitigation M_i for which mrEffect is less than -1.
- **If** $mrEffect[M_i][R_j]$ between $(-1, 0)$ **then** $rLikelihood[R_j]$ is initialized is increased to the value of the currrent risk likelihood plus the magnitude of the effect. In short, $rLikelihood[R_j] = Minimum(rlikelihood[R_j] - m[M_i] * mrEffect[M_i][R_j])$. Note that the "-" sign becomes an addition because the mrEffect is negative in this case. Also

note that the maximum rLikelihood is enforced to be 1 by the "Minimum" function.

- **For all** $mrEffect[M_i][R_j]$ **then** $rLikelihood[R_j]$ is set to the product of it's current value by the sequential factors of $(1 - m[M_i] * mrEffect[M_i][R_j])$ where M_i represents the mitigations that have an effect on the given risk R_j, and $m[M_i]$ represents the selection of the mitigation M_i such that if it is equal to 1 when the mitigation has been selected for implementation or it is equal to 0 when it is not selected. In short, $rLikelihood[R_j] = rAPL[R_j] * \prod(1 - mrEffect[M_i][R_j])$ for all selected mitigations M_i that have an effect on risk R_j.

The array "mrEffect" is printed inside the "SetupModel()" function and the function ends. The other two arrays generated, namely rLikelihood and rAggrevatedImpact, are stored in memory and printed out inside the "model(float m[])" function after aggravated impacts are initialized to 1 and likelihoods to the risk a-priori likelihoods (rAPL) assigned in the setup funtion. The "model(float m[])" function therefore is printed in the following order:

- Risk aggravated impacts initialized to 1
- Risk likelihoods initialized to a-priori likelihoods
- Risk aggravated impacts from the rAggrevatedImpact array generated during setup
- Risk likelihoods from the rLikelihood array generated during setup

Here it is the pseudo code for printing this first portion of the model function:

```
print model funtion header
for REQ in requirements
{
  print "rAggrevatedImpact[REQ] = 1"
}
for REQ in requirements
{
  print "rLikelihood[REQ] = rAPL[REQ]"
}
```

The remaining portion of the model funtion consists of printing impacts of risks on requirements attainment proportions (oAtRiskProp[REQ]) and requirements attainment (oAttainment[REQ]) for each requirement, the total mitigation cost (costTotal), and the total requirements attainment (attTotal).

The impacts of risks on requirement attainment proportion is a per requirement calculation where the product of the risk likelihood, aggravated impact, and the impact of each risks is added. In short,

$$oAtRiskProp[O_i] = \sum(rlikelihood[R_j] * rAggrevatedImpact[R_j] * roImpact[R_j][O_i]);$$

for all risks R_j that have an impact on requirement O_i.

The requirement attainment for each requirement is then calculated as the requirement priority (oWeight) for the given requirement times the inverse proportion of the impact of risks on the attainment for the given requirement (roAtriskProp). In short,

$$oAttainment[O_i] = oWeight[O_i] * (1 - Minimum(1, oAtRiskProp[O_i])$$

The total attainment is printed next and it is the result of adding the requirement attainment for each requirement:

For each REQ in requirements **Do**
{
  attTotal = $\sum$(oAttainment[REQ])
}

Lastly, the total cost is calculated by adding the costs of the implemented mitigations. This is achieved by the following pseudocode:

For each MIT in mitigations **Do**
{
  costTotal = $\sum(m[MIT] *$ mCost[MIT])
}

4. SOURCE CODE

The following is the complete listing of the code.

```
#!/bin/bash

declare -a mCost
declare -a oWeight
declare -a rAPL
declare -a roImpactR
declare -a roImpactO
declare -a roImpactV
declare -a mrEffectM
declare -a mrEffectR
declare -a mrEffectV
declare -a rAggImpV
declare -a minVal
declare -a rAggImpT
declare -a mrEP
declare -a oARP

RANDOM=$$$(date +%s)

MCOUNT=5
OCOUNT=4
RCOUNT=3
PHASES=5

### FUNCTION: HELP ###
usage()
{
cat << EOF
usage: $0 [OPTIONS]

This script randomly generates ddp instances according \
to the arguments specified.

OPTIONS:
   -h      Show this message
   -m      number of mitigations
   -o      number of objectives
   -r      number of requirements
EOF
}


### ARGUMENT HANDLING ###
while getopts  "hm:o:r:" OPTION
do
     case $OPTION in
m) MCOUNT=$OPTARG;;
o) OCOUNT=$OPTARG;;
r) RCOUNT=$OPTARG;;
h) usage; exit;;
?) usage; exit;;
     esac
done

#### FILE HEADER AND FOOTER
headSU="#include \"model.h\"\n\n#define M_COUNT $MCOUNT\n
        #define O_COUNT $OCOUNT\n#define R_COUNT $RCOUNT\n\n"
struct="struct ddpStruct{\n\tfloat oWeight[O_COUNT+1];\n
        \tfloat oAttainment[O_COUNT+1];\n
        \tfloat oAtRiskProp[O_COUNT+1];\n
        \tfloat rAPL[R_COUNT+1];\n
        \tfloat rAggrevatedImpact[R_COUNT+1];\n
        \tfloat rLikelihood[R_COUNT+1];\n
        \tfloat mCost[M_COUNT+1];\n
        \tfloat roImpact[R_COUNT+1][O_COUNT+1];\n
        \tfloat mrEffect[M_COUNT+1][R_COUNT+1];\n};
        \n\nddpStruct *ddpData;\n\nvoid setupModel(void)\n{\n
        \tddpData = (ddpStruct *) malloc(sizeof(ddpStruct));"
headM="}\n\nvoid model(float *cost, float *att, float m[])\n{\n
        \tfloat costTotal, attTotal;\n"
tail="\n\t*cost = costTotal;\n\t*att = attTotal;\n}\n"


genRandInt(){
    min=$1
    max=$2
    val=$[ ( ($RANDOM % ( ($max+1) - $min ) ) + $min ) ]
    echo $val
}

genRandFloat(){
    min=$1
    max=$2
    factor=100
    min=$[ ( $min * $factor ) ]
    max=$[ ( $max * $factor ) ]
    val=$( genRandInt $min $max )
    echo "scale=3; $val / $factor" | bc
}

genArrays(){
```

```
    for (( i=1; i<=$MCOUNT; i++ )){
val=$( genRandInt 0 32000 )
mCost[$i]=$val
    }

    for (( i=1; i<=$OCOUNT; i++ )){
val=$( genRandInt 1 100 )
oWeight[$i]=$val
    }

    roImpactSize=0
    for (( i=1; i<=$RCOUNT; i++ )){
rAPL[$i]=1
rAggImpV[$i]=1
for (( j=1; j<=$OCOUNT; j++ )){
    happens=$( genRandInt 0 1 )
    if [ $happens -eq 0 ]; then
val=$( genRandFloat 0 1 )
val=$( printf %1.1f $val )
roImpactSize=$[ ( $roImpactSize + 1 ) ]
roImpactR[$roImpactSize]=$i
roImpactO[$roImpactSize]=$j
roImpactV[$roImpactSize]=$val
oARP[$j]="${oARP[$j]} + (ddpData->rLikelihood[$i] *
         ddpData->rAggrevatedImpact[$i] *
         ddpData->roImpact[$i][$j]) "
    fi
}
    }

    currPhase=1
    minValSize=0
    mrEffectSize=0
    for (( i=1; i<=$MCOUNT; i++ )){
for (( j=1; j<=$RCOUNT; j++ )){
    happens=$( genRandInt 0 1 )
    if [ $happens -eq 0 ]; then
val=$( genRandFloat 0 1 )
negative=$( genRandInt 0 20 )
if [ $negative -gt 19 ]; then
    val=$( echo "scale=2; $val * -1" | bc )
    aggravated=$( genRandInt 0 4 )
    if [ $aggravated -gt 3 ]; then
val=$( echo "scale=2; $val - 1" | bc )
rAggImpT[$j]="${rAggImpT[$j]} * (1 - m[$i] *
             (1 + ddpData->mrEffect[$i][$j]))"
    else
minValSize=$[ ( $minValSize + 1 ) ]
minVal[$minValSize]="ddpData->rLikelihood[$j]=
                     minValue(1, (ddpData->rLikelihood[$j] -
                     m[$j] * ddpData->mrEffect[$i][$j]));"
    fi
else
    ### SPLIT array in phases
    incPhase=$( genRandInt 0 4 )
    if [ $incPhase -gt 3 ]; then
currPhase=$[ ( $currPhase + 1 ) ]
    fi
    case $currPhase in
1) mrEP1[$j]="${mrEP1[$j]} *
              (1 - m[$i] *
              ddpData->mrEffect[$i][$j])";;
2) mrEP2[$j]="${mrEP2[$j]} *
              (1 - m[$i] *
              ddpData->mrEffect[$i][$j])";;
3) mrEP3[$j]="${mrEP3[$j]} *
              (1 - m[$i] *
              ddpData->mrEffect[$i][$j])";;
4) mrEP4[$j]="${mrEP4[$j]} *
              (1 - m[$i] *
              ddpData->mrEffect[$i][$j])";;
?) mrEP5[$j]="${mrEP5[$j]} *
              (1 - m[$i] *
              ddpData->mrEffect[$i][$j])";;
    esac
fi
val=$( printf %1.2f $val )
mrEffectSize=$[ ( $mrEffectSize + 1 ) ]
mrEffectM[$mrEffectSize]=$i
mrEffectR[$mrEffectSize]=$j
mrEffectV[$mrEffectSize]=$val
    fi
}
    }
}

printFile(){
    genArrays

    echo -e $headSU
    echo -e $struct

    for (( i=1; i<=$MCOUNT; i++ )){
echo -e "\tddpData->mCost[$i]= ${mCost[$i]};"
    }
    for (( i=1; i<=$RCOUNT; i++ )){
echo -e "\tddpData->rAPL[$i]= ${rAPL[$i]};"
    }
    for (( i=1; i<=$OCOUNT; i++ )){
echo -e "\tddpData->oWeight[$i]= ${oWeight[$i]};"
    }
    for (( i=1; i<=$roImpactSize; i++ )){
echo -e "\tddpData->roImpact[${roImpactR[$i]}]
        [${roImpactO[$i]}]= ${roImpactV[$i]};"
    }
    for (( i=1; i<=$mrEffectSize; i++ )){
echo -e "\tddpData->mrEffect[${mrEffectM[$i]}]
        [${mrEffectR[$i]}]= ${mrEffectV[$i]};"
    }

    echo -e $headM

## PRINT rAggrevatedImpact initialization
    for (( i=1; i<=$RCOUNT; i++ )){
echo -e "\tddpData->rAggrevatedImpact[$i] =
        ${rAggImpV[$i]};"
    }

## PRINT rLikelihood initialization
    for (( i=1; i<=$RCOUNT; i++ )){
echo -e "\tddpData->rLikelihood[$i] = ddpData->rAPL[$i];"
    }

## PRINT rAggrevatedImpact when mrEffect [-1..0]
    for (( i=1; i<=$minValSize; i++ )){
echo -e "\t${minVal[$i]}"
    }

## PRINT rAggrevatedImpact when mrEffect < -1
    for (( i=1; i<=$RCOUNT; i++ )){
if [ -n "${rAggImpT[$i]}" ]; then
    echo -e "\tddpData->rAggrevatedImpact[$i] =
            ddpData->rAggrevatedImpact[$i] ${rAggImpT[$i]};"
fi
    }

## PRINT rLikelihood when mrEffect [0..1]
# First, split the likelihoods into PHASES
    echo -e "\n\t/* Mitigations Effects on Risk likelihoods */\n
            \t/*Phase I mitigation effects*/"
    for (( i=1; i<=$RCOUNT; i++ )){
if [ -n "${mrEP1[$i]}" ]; then
    echo -e "\tddpData->rLikelihood[$i] =
            ddpData->rLikelihood[$i] ${mrEP1[$i]};"
fi
    }
    echo -e "\t/* Phase II mitigation effects*/"
    for (( i=1; i<=$RCOUNT; i++ )){
if [ -n "${mrEP2[$i]}" ]; then
    echo -e "\tddpData->rLikelihood[$i] =
            ddpData->rLikelihood[$i] ${mrEP2[$i]};"
fi
    }
    echo -e "\t/* Phase III mitigation effects*/"
    for (( i=1; i<=$RCOUNT; i++ )){
if [ -n "${mrEP3[$i]}" ]; then
    echo -e "\tddpData->rLikelihood[$i] =
            ddpData->rLikelihood[$i] ${mrEP3[$i]};"
fi
    }
    echo -e "\t/* Phase IV mitigation effects*/"
    for (( i=1; i<=$RCOUNT; i++ )){
if [ -n "${mrEP4[$i]}" ]; then
    echo -e "\tddpData->rLikelihood[$i] =
            ddpData->rLikelihood[$i] ${mrEP4[$i]};"
fi
    }
    echo -e "\t/* Phase V mitigation effects*/"
    for (( i=1; i<=$RCOUNT; i++ )){
if [ -n "${mrEP5[$i]}" ]; then
    echo -e "\tddpData->rLikelihood[$i] =
            ddpData->rLikelihood[$i] ${mrEP5[$i]};"
fi
    }

## PRINT oAtRiskProp
    echo -e "\n\t/* Risk impacts on objective
            attainment proportions */"
    for (( i=1; i<=$OCOUNT; i++ )){
if [ -n "${oARP[$i]}" ]; then
    echo -e "\tddpData->oAtRiskProp[$i] = 0${oARP[$i]};"
fi
    }

## PRINT oAttainment
    echo -e "\n\t/* Objective Attainments */"
    for (( i=1; i<=$OCOUNT; i++ )){
echo -e "\tddpData->oAttainment[$i] =
        ddpData->oWeight[$i] *
        (1 - minValue(1, ddpData->oAtRiskProp[$i]));"
    }

## PRINT attTotal
    for (( i=1; i<=$OCOUNT; i++ )){
attTotal="$attTotal + ddpData->oAttainment[$i]"
    }
    echo -e "\tattTotal = 0$attTotal;"

## PRINT costTotal
    for (( i=1; i<=$MCOUNT; i++ )){
costTotal="$costTotal + m[$i] * ddpData->mCost[$i]"
    }
    echo -e "\tcostTotal = 0$costTotal;"

    echo -e $tail
}
printFile
```

5. REFERENCES

[1] M. Feather, S. Cornford, K. Hicks, J. Kiper, and T. Menzies. Application of a broad-spectrum quantitative requirements model to early-lifecycle decision making. *IEEE Software*, 2008. Available from `http://menzies.us/pdf/08ddp.pdf`.