

# DoDAF Methods for Software Engineers

C.W. Perr and Dr. John A. Hamilton Jr.

## Abstract

### Improving DoDAF for Software Engineers

C.W. Perr Auburn University  
 cwp0002@tigermail.auburn.edu

Dr. John A. Hamilton, Jr. Auburn University  
 hamilton@auburn.edu

26 August 2011

#### Abstract

At the last Annual SERC Research Review (ASRR) serious concerns were brought up about the current practices regarding the Department of Defense Architecture Framework (DoDAF). In our own work with DoDAF we noted that the overhead is high, and depending on the current stage of the project, the value gained might be limited. At the same time DoDAF presents itself as a possible solution to making well-informed management and design decisions, and as such is mandated in numerous Department of Defense (DoD) projects. At Auburn University we wanted to know how to make DoDAF useful on software intensive projects and sought to create a set of best practices for using DoDAF for software engineering.

To develop these best practices we generated DoDAF viewpoints through-out the design and implementation of a project that we were working on: The Counter Improved Explosive Device (IED) Medical Technology software trainer, or CIMT for short. This project utilized the Unity Game Engine to develop a virtual trainer for sharpening battlefield first-responders' triage and treatment techniques.

In generating DoDAF viewpoints on this project we were able to create a test case where we were using complex off the shelf commercial software, our own custom scripts and 3d models purchased through web retailers. To model the interplay between these closed-source and open-source assets in a way that was meaningful we needed to combine office software that is typically used with custom tools to display information in a natural manner. For these cases custom python scripts and Orange were used.

In our application, while all views were generated, it was noted that a great deal of overlap existed. As such we were able and encouraged to select the viewpoints which were the most useful to us and to remove the ones that were not from the final DoDAF product. The presentation tool Prozi was then used to display those views because of its inherent ability to view documents from different levels of abstraction.

The main concepts that we learned to use to make DoDAF a success in our project were to: allow the viewpoints to be constantly updated, to view the documents as a management priority, and to allow for automation wherever possible. In the future we hope to focus our work more into generating these documents from source code through automation.

### Counter IED Medical Trainer (CIMT)



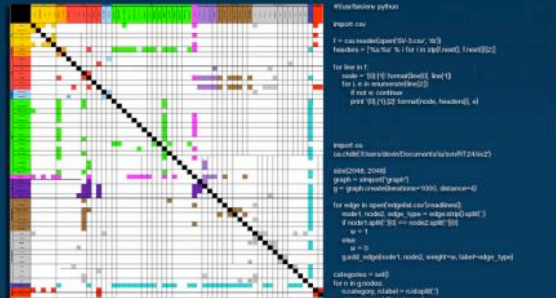
## Automation

The importance of automation cannot be understated. In the cases where our team was able to apply automation the cost was usually minimal yet gave us the best results.

After importing the SV-3 into a comma separated values file we were able to create a directed graph, which was then fed into Nodebox using a custom script (buildgraph.py) to create an edge list. This list was then fed into NodeBox and Inkscape to create an SV-2 which is easier to read for both the software developers and managers.

By letting the team develop their own methods for DoDAF view creation we hit upon methods which were faster and created higher quality documents than using common office software.

These custom methods produced better architecture and eliminated non-value added efforts.



```
#!/usr/bin/python
import csv
import sys

# In case we're running from a script
headers = ['Name', 'ID', 'Type', 'ParentID']

for line in sys.stdin:
    node = line.strip().split(',')
    # not a custom
    print '%s,%s,%s,%s,%s' % (node[0], node[1], node[2], node[3], node[4])

import os
import sys
import os.path

# Create a directory for the graph
graph_dir = 'graph'
if not os.path.exists(graph_dir):
    os.makedirs(graph_dir)

# Create a file for the graph
graph_file = 'graph.csv'
if not os.path.exists(graph_file):
    open(graph_file, 'w').close()

# Write the graph to the file
with open(graph_file, 'a') as f:
    for line in sys.stdin:
        node = line.strip().split(',')
        # not a custom
        print '%s,%s,%s,%s,%s' % (node[0], node[1], node[2], node[3], node[4])

categories = set()
for n in graph.nodes:
    categories.add(n.category)

nodes = []
for n in graph.nodes:
    nodes.append(n)

print 'Nodes: %s' % str(nodes)
print 'Edges: %s' % str(graph.edges)
print 'Categories: %s' % str(categories)
```



## Results

