Fraunhofer
CESE

Architectural Analysis of Dynamically Reconfigurable Systems

Technical Presentation

Presenter: Dharma Ganesan
Principal Investigator (PI): Dr. Mikael Lindvall, CESE
NASA POC: Sally Godfrey, GSFC
Team members:
Dr. Chris Ackermann, Dr. Arnab Ray, Lyly Yonkwa (CESE)
GMSEC, CFS, RBSP
In collaboration with Fraunhofer Institute for Experimental Software Engineering (IESE)

Center for Experimental Software Engineering

Background

- Many NASA projects use flexible architecture styles for
  - creating loosely coupled systems
  - minimizing future software change
- Examples of such systems:
  - Goddard Mission Services Evolution Center (GMSEC)
    - A reusable framework for ground systems
  - Core Flight Software (CFS)
    - A reusable framework for flight systems

Problem

- Increased flexibility of architectural styles decrease analyzability
- Behavior emerges and varies depending on the configuration
- Does the resulting system run according to the intended design?
- What architectural decisions impede or facilitate testing?

Top Down Approach

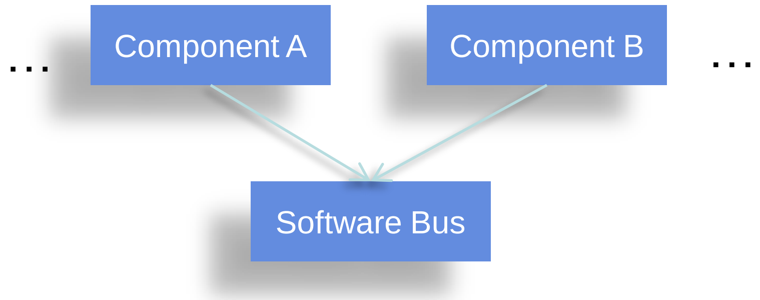
- Architecture analysis
  - focusing on critical components' behavior data
  - visualizing architecture relevant events
  - drilling down to details as necessary
- Detect defects and deviations
  - modeling, comparing planned vs. actual behavior
- Architecture and its testability

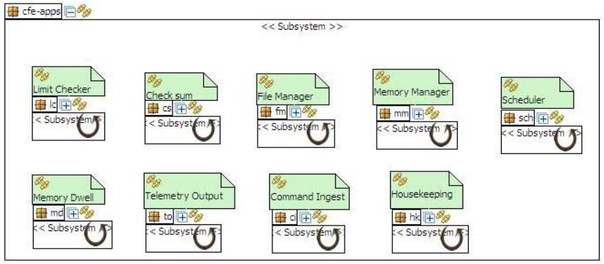
Currently Targeted Projects: GMSEC and CFS

- Reusable framework for ground and flight systems
- GMSEC and CFS systems are running at FC-MD
- Confirmed defects/violations reported in several papers

➢ Some example results

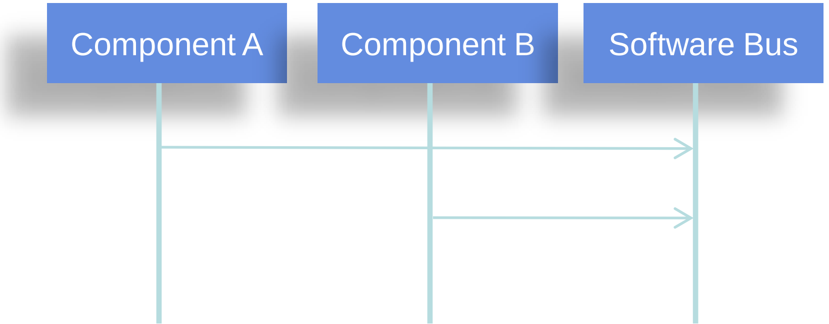
Analyzing Software Architectures





No static dependencies!
→static analysis is not sufficient

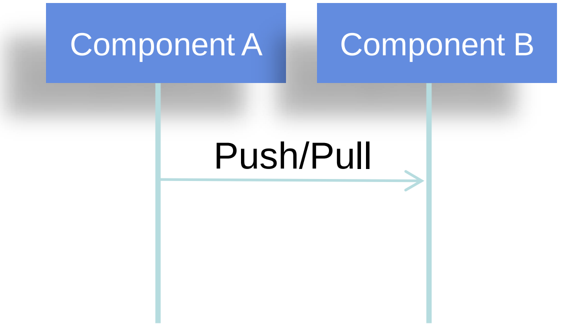
Dynamic Save



Run-time Events difficult to analyze because
There are too many low level events

Goal

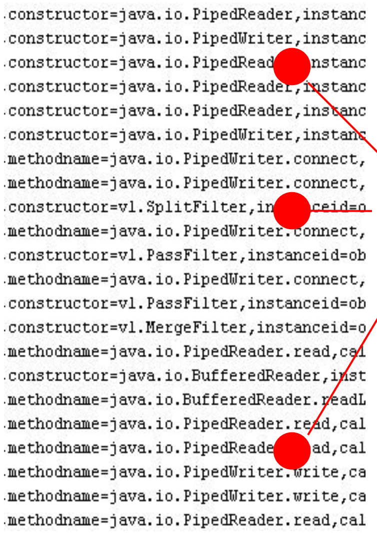
New tool



New tool can detect architecture relevant events and hide irrelevant information

Analyzing Runtime Events

- Problems
  - – different events are of interest
  - – events can occur in any order
  - – huge number of events
  - – range between events might be very large



```
.constructor=java.io.PipedReader,instanc
.constructor=java.io.PipedWriter,instanc
.constructor=java.io.PipedRead[illegible]nstanc
.constructor=java.io.PipedReader,instanc
.constructor=java.io.PipedReader,instanc
.constructor=java.io.PipedWriter,instanc
.methodname=java.io.PipedWriter.connect,
.methodname=java.io.PipedWriter.connect,
.constructor=vl.SplitFilter,in[illegible]ceid=o
.methodname=java.io.PipedWriter.connect,
.constructor=vl.PassFilter,instanceid=ob
.methodname=java.io.PipedWriter.connect,
.constructor=vl.PassFilter,instanceid=ob
.constructor=vl.MergeFilter,instanceid=o
.methodname=java.io.PipedReader.read,cal
.constructor=java.io.BufferedReader,inst
.methodname=java.io.BufferedReader.readL
.methodname=java.io.PipedReader.read,cal
.methodname=java.io.PipedReade[illegible]ad,cal
.methodname=java.io.PipedWriter.write,ca
.methodname=java.io.PipedWriter.write,ca
.methodname=java.io.PipedReader.read,cal
```

points of interest

Solutions: Goal-oriented data collection and a pattern recognition engine

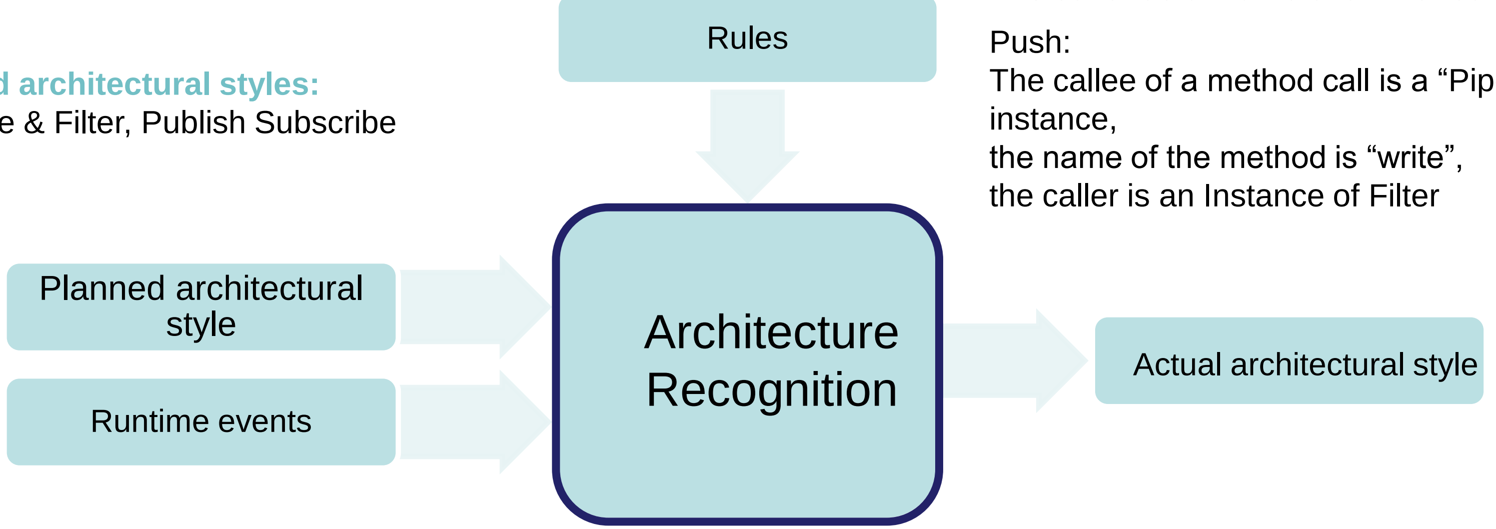
Actual Architecture Recognition

Planned architectural styles:
E.g. Pipe & Filter, Publish Subscribe

Rules:

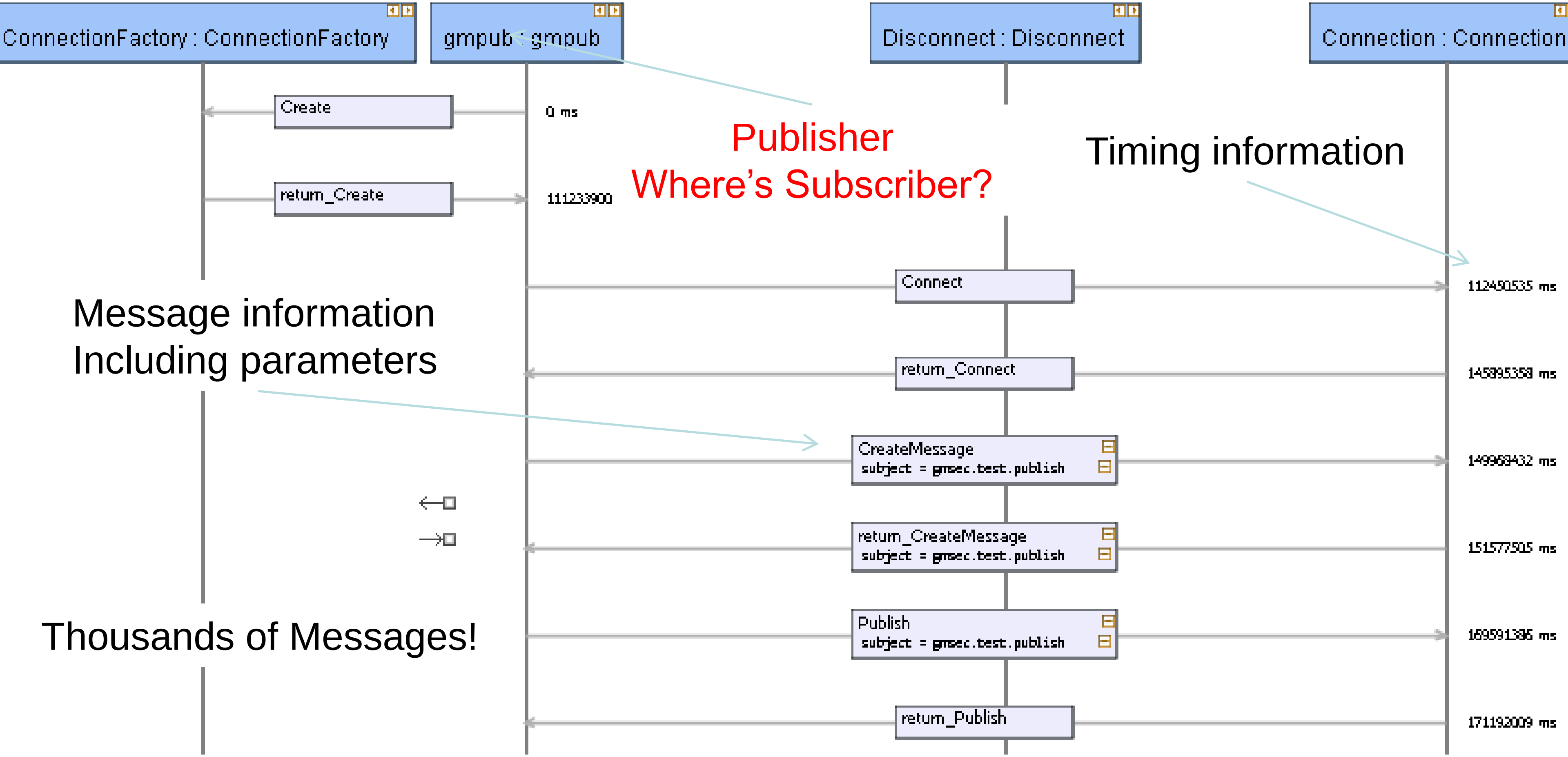
Filter:
The constructor name of a Filter contains "Filter"

Push:
The callee of a method call is a "PipedWriter" instance,
the name of the method is "write",
the caller is an Instance of Filter



Runtime events:
init,timestamp=1264620606308,constructor=v1.MergeFilter,instanceid=obj578ceb
call,timestamp=1264620606317,methodname=java.io.PipedReader.read,callee=obj9ed927,caller=objfa7e74,argument=null

GMPUB in Dynamic SAVE



Publisher
Where's Subscriber?

Timing information

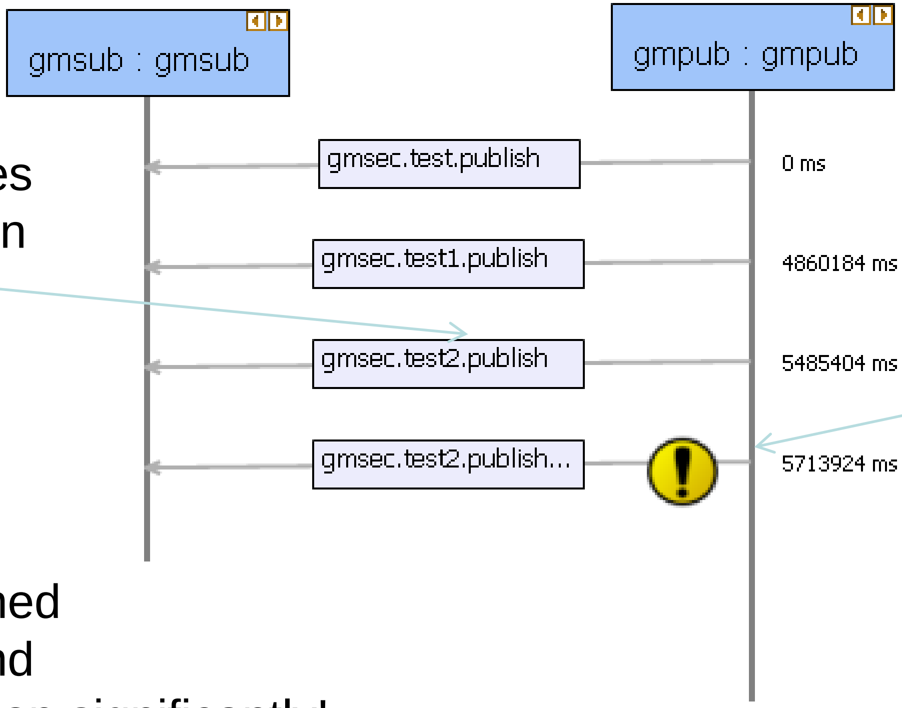
Message information
Including parameters

Thousands of Messages!

This diagram was automatically created by Dynamic SAVE
using run time information from GMSEC

Problem: Much information, but GMSUB component that receives messages missing!

Sample output from new approach



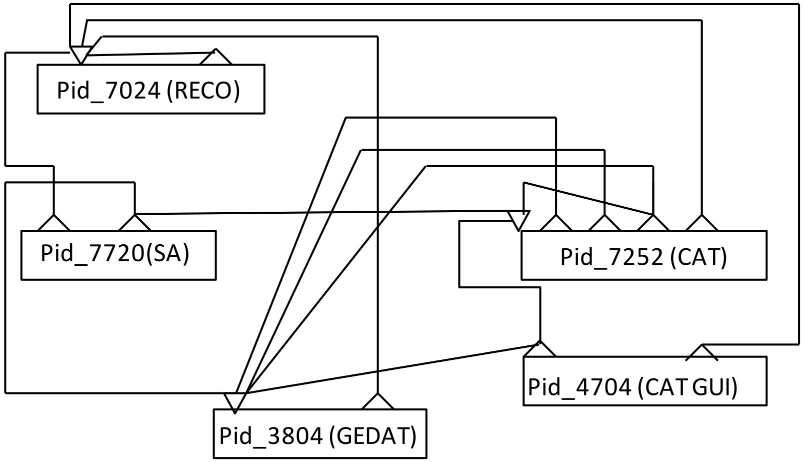
Only critical messages
Visible, all else hidden

Unexpected
Duplicate
message!

Pattern engine matched
pairs of messages and
reduced the information significantly!

This diagram will be automatically created by the new approach
using the same run time information from GMSEC

Sample output from new approach …

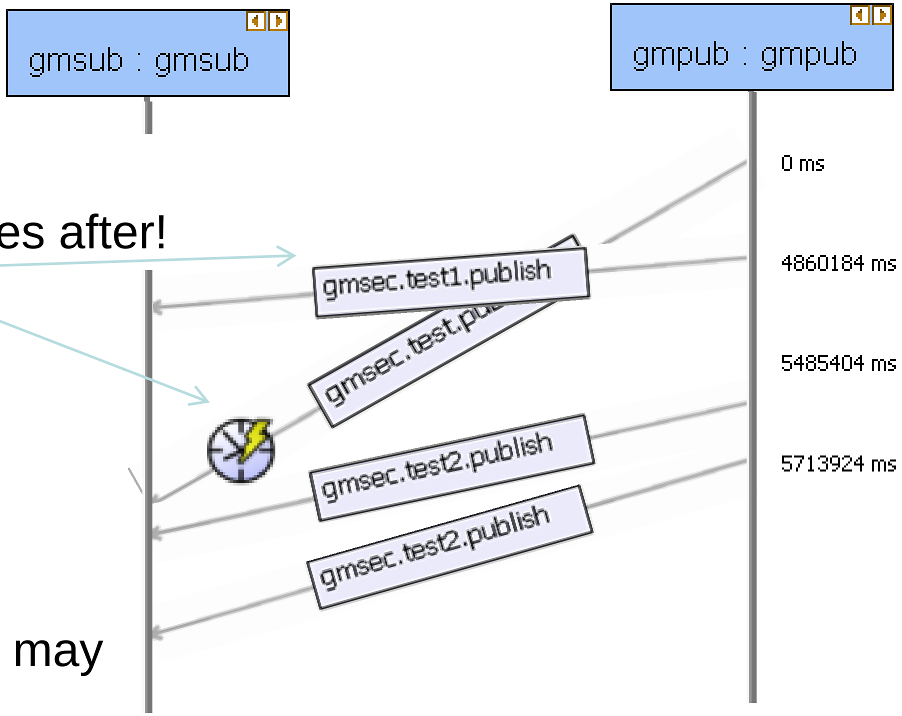


△ Connection port for publishing to the software bus

▽ Connection port for subscribing to the software bus

This diagram will be semi-automatically created by the new approach using the same run time information from GMSEC

Taking message timing delays into account



Timing is off!
Sent before but arrives after!

The slopes indicate message delays that may Impact behavior

This diagram will be automatically created by the new approach using the same run time information from GMSEC

Architecture and Testability – CFS Examples

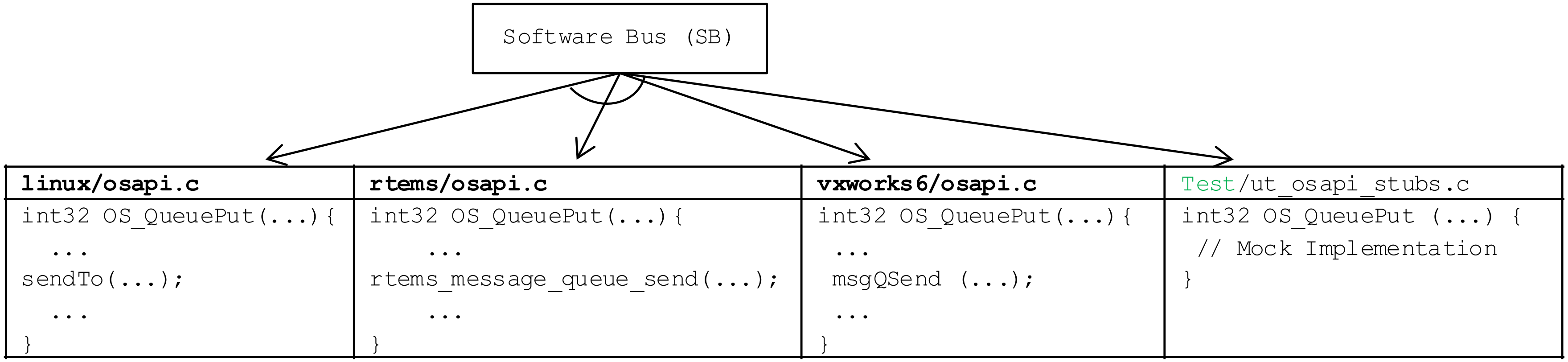
- We analyze the CFS architecture and its unit testing architecture

- Focus of the analysis:
  - What architectural decisions impede or facilitate testing?

Some Recommendations for improved testability

- Modules should be programmed to abstract interfaces
  - mock implementations of interfaces for unit testing
- Some internal details of modules should be public – cannot “hide” everything
- Avoid using the same return code of functions for different scenarios

Abstract Interfaces and Testability – CFS example



Software Bus (SB)

linux/osapi.c

```
int32 OS_QueuePut(...){
   ...
sendTo(...);
   ...
}
```

rtems/osapi.c

```
int32 OS_QueuePut(...){
      ...
rtems_message_queue_send(...);
      ...
}
```

vxworks6/osapi.c

```
int32 OS_QueuePut(...){
  ...
 msgQSend (...);
  ...
}
```

Test/ut_osapi_stubs.c

```
int32 OS_QueuePut (...) {
  // Mock Implementation
}
```

Open some internal details – CFS example

```
int32 CFE_ES_LoadLibrary(char *EntryPoint,  char *LibName, …) {
   boolean LibSlotFound = FALSE;
   for ( i = 0; i < CFE_ES_MAX_LIBRARIES; i++ ) {
      if ( CFE_ES_Global.LibTable[i].RecordUsed == FALSE ) {
          LibSlotFound = TRUE;
          break;
      }
   }
   if(LibSlotFound == FALSE) return CFE_ES_ERR_LOAD_LIB;
}
```

```
/* Test for loading more than max number of libraries */
 for (j= 0; j < CFE_ES_MAX_LIBRARIES; j++) {
        CFE_ES_Global.LibTable[j].RecordUsed = TRUE;
 }
 Return = CFE_ES_LoadLibrary("EntryPoint","LibName“, …);
 UT_Report(Return == CFE_ES_ERR_LOAD_LIB, "CFE_ES_LoadLibrary",
                        "No free   library slots");
```

Summary and Next Steps

- We’re building a new approach that
  - helps understand, visualize, and validate software systems that use loosely coupled architecture styles
  - helps evaluating testability of the architecture
- Next steps
  - refine software tools and method, apply also to other NASA systems

Acronyms

- AFRL – Air Force Research Laboratory
- APL – Applied Physics Laboratory
- ARC – Ames Research Center
- CESE – Center for Experimental Software Engineering
- cFE – core Flight Executive
- CFS – Core Flight Software

Acronyms (2)

- CHIPS - Cosmic Hot Interstellar Plasma Spectrometer
- CLARREO - Climate Absolute Radiance and Refractivity Observatory
- COTS – Commercial Off-The-Shelf
- DSILCAS – Distributed System Integrated Lab Communications Adapter Set
- Dyn-SAVE – Dynamic SAVE

Acronyms (3)

- GLAST - Gamma-ray Large Area Space Telescope
- GMSEC – Goddard Mission Services Evolution Center
- GOTS – Government Off-The-Shelf
- GPM - Global Precipitation Measurement
- GSFC – Goddard Space Flight Center
- IV& V – Independent V & V

Acronyms (4)

- JSC – Johnson Space Center
- LADEE - Lunar Atmosphere and Dust Environment Explorer
- LDCM - Landsat Data Continuity Mission
- LRC - Langley Research Center
- LRO - Lunar Reconnaissance Orbiter
- MMOC – Multi-Mission Operations Center
- MMS - Magnetospheric MultiScale

Acronyms (5)

- MSFC - Marshall Space Flight Center
- RBSP – Radiation Belt Storm Probes
- SAVE – Software Architecture Visualization and Evaluation
- SDO – Solar Dynamics Observatory
- TRMM – Tropical Rainfall Measuring Mission
- V & V – Verification and Validation