Code Risk Estimation for OpNav

June 2021

Joseph Kostial

Jay Marchetti

Nicholas Reimer

Michael Riley

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Copyright 2021 Carnegie Mellon University.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

Carnegie Mellon® and CERT® are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

DM21-0486

BLUF

Bottom Line Up Front

NASA OpNav 1.5 Code Risk Estimation

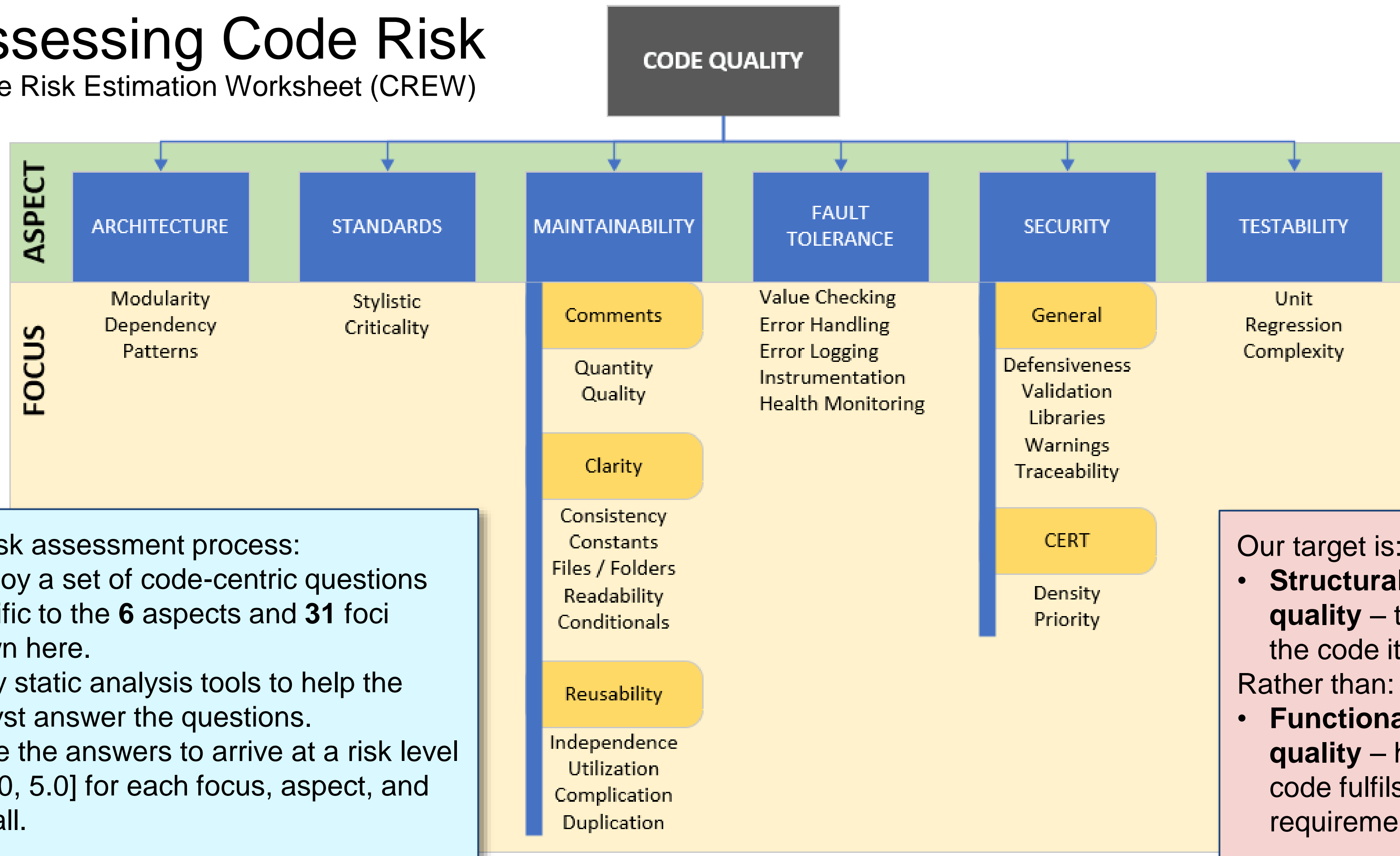
CREW Total Risk Score = **2.50**

MAINTAIN	IMPROVE
• Good module & code layout consistency and file & folder organization	• Reduce the number of conditional compilation flags in the code and / or make them run-time traceable
• Uniform adherence to stylistic NASA Flight Software Branch C Coding Standard	• Adopt and adhere to a critical coding standard such as MISRA or JSF++ and enforce it via static analysis
• Consistent error handling and logging	• Eliminate dynamic memory allocation after boot-time initialization
• Solid unit testing and (evidence of) automated regression testing	• Eliminate duplicated source files and copy / pasted code sections
• Good run-time code instrumentation & monitoring	• Implement traceability of data & executable files with anti-tamper verification at boot-time
• Excellent accuracy, level of, and clarity of code commenting	• Reduce the number of cyclic and bi-directional dependencies
• Low density of secure coding issues	• Reduce the depth of control structure & call nesting

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Assessing Code Risk

Code Risk Estimation Worksheet (CREW)



Code risk assessment process:

- Employ a set of code-centric questions specific to the **6** aspects and **31** foci shown here.
- Apply static analysis tools to help the analyst answer the questions.
- Score the answers to arrive at a risk level in [1.0, 5.0] for each focus, aspect, and overall.

Our target is:

- **Structural code quality** – the quality of the code itself.

Rather than:

- **Functional code quality** – how well the code fulfils mission requirements.

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Overview

- OpnNav code quality is the best that SEI has reviewed to date.
  - The CREW total risk score of **2.50** indicates a solid code design and implementation.
  - Fault tolerance and testability are stand out characteristics – both imply lowered probability of latent defects.
  - SEI recommends improvements targeting reusability and general security to further enhance OpNav code quality.
  - Incorporating elements of a critical system coding standard would benefit overall OpNav quality.

CODE RISK ESTIMATION WORKSHEET: OpNav

ASPECT RISK	ASPECT /	Focus	MITIGATION
2.4	ARCHITECTURE /		
	2.2	Modularity	High level of code modularity and module cohesiveness
	3.5	Dependency	Low level of dependencies, especially cyclic (DSM, dependency graphs, etc)
	1.6	Patterns	High level of adherence to a known architectural pattern (layered, client-server, etc)
2.6	STANDARDS /		
	1.6	Stylistic	High level of adherence to a chosen internal or published stylistic coding standard
	3.5	Criticality	High level of adherence to critical-system coding guidelines (MISRA, CERT, etc)
2.7	MAINTAINABILITY /		
Comments	1.2	Quantity	High level of commenting excluding boilerplates
	1.9	Quality	High quality of commenting that clarifies the code's intent
Clarity	1.2	Consistency	High level of code stylistic consistency (naming, source layout, brace usage, etc)
	2.7	Constants	High level of use of symbolic constants
	1.5	Files / Folders	High level of source file and folder naming consistency and organization
	3.1	Readability	High level of overall code readability
	4.0	Conditionals	Low level of conditional compilation
Reusability	1.5	Independence	High level of machine, OS, database, and compiler independence
	3.8	Utilization	Low level of commented-out or dead code, and unused variables
	4.1	Complication	Low level of complicated implementation constructs
	5.0	Duplication	Low level of duplicated (copy / pasted) code
1.6	FAULT TOLERANCE /		
	2.0	Value Checking	High level of return and parameter value checking
	1.0	Error Handling	Existence of a systematic error handling mechanism
	1.9	Error Logging	Existence of an error / fault logging mechanism
	1.1	Instrumentation	Existence of run-time statistics and timing instrumentation
	2.1	Health Monitoring	Effective use of watchdog and system health monitoring
3.5	SECURITY /		
General	3.0	Defensiveness	High level of defensive / secure coding practices
	3.8	Validation	High level of validation of untrusted inputs / interfaces
	2.5	Libraries	Low level of use of potentially unsafe library calls
	3.3	Warnings	Evidence that code compiles with no (or few) warnings
	5.0	Traceability	Effective use of executable and parameter file signatures / anti-tamper measures
CERT	1.5	Density	Low density of secure coding issues as per SEI CERT Secure Coding Rules
	2.4	Priority	Low number of high-priority secure coding issues as per SEI CERT Secure Coding Rules
1.9	TESTABILITY /		
	1.4	Unit	Evidence of unit testing, especially automated (test cases, test harness, etc)
	1.4	Regression	Evidence of regression testing, especially automated
	2.8	Complexity	Low valued cyclomatic complexity (CC1) and path count
	2.50		TOTAL RISK

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Architecture

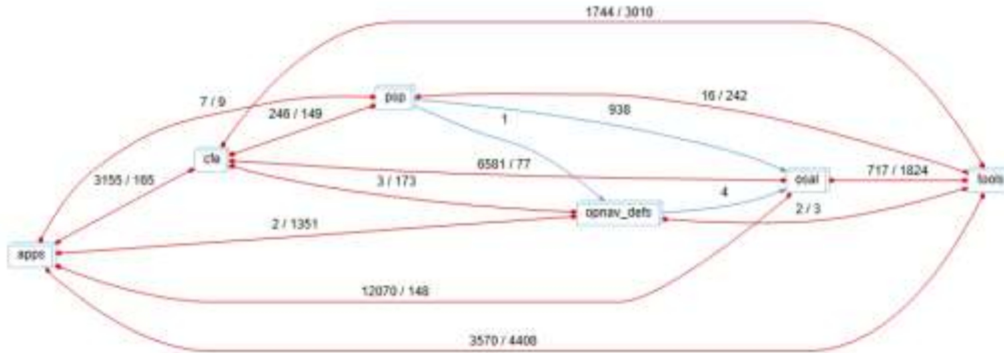
CODE RISK ESTIMATION WORKSHEET: OpNav			
ASPECT RISK	ASPECT / Focus		MITIGATION
2.4	ARCHITECTURE /		
	2.2	Modularity	High level of code modularity and module cohesiveness
	3.5	Dependency	Low level of dependencies, especially cyclic (DSM, dependency graphs, etc)
	1.6	Patterns	High level of adherence to a known architectural pattern (layered, client-server, etc)

- The OpNav code base shows adherence to a pre-planned architecture intended for reuse.
  - Modularity and cohesiveness are strong. System stability (94.8%) is high indicating that changes would leave most code unaffected even though numerous static global variables are used.
  - Dependencies among elements are average in number but show many bi-directional and some cyclic dependencies.
  - The dependency structure matrix (DSM) shows a non-strictly layered architectural pattern. The layers present consistent application programming interfaces (API).

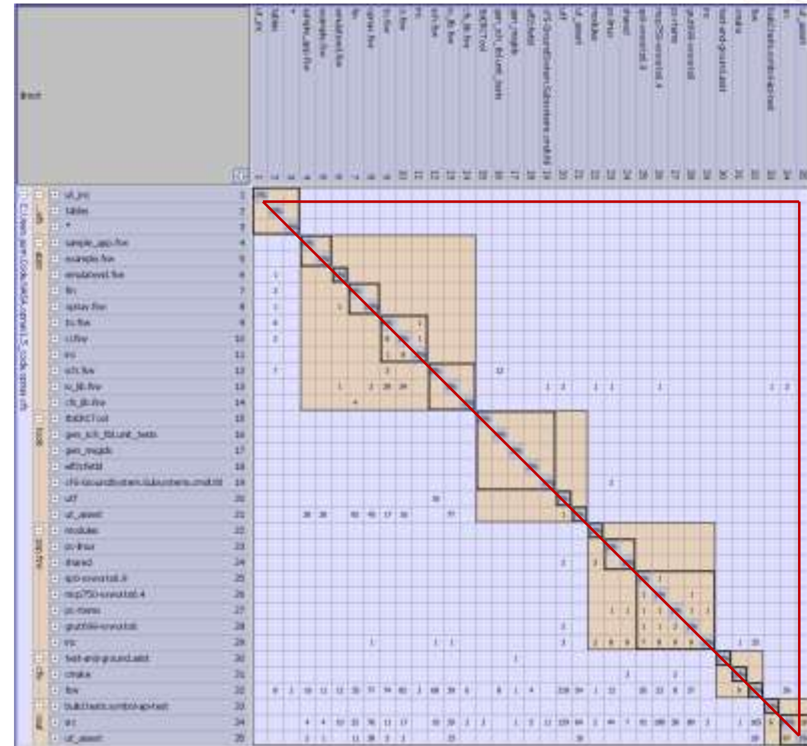
[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Architecture

Detail



- The top-level dependency graph shows mostly bi-directional (red) dependencies between primary components.
- Architectural metrics for OpNav include:
  - System stability = 94.8%
  - System cyclicality = 9.5%
  - Architectural complexity = 0.52
  - Atom count = 955 and Edge count = 5,427



- The mid-level DSM shows non-strict layered components with some inter- and intra-component cyclic dependencies.

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Standards

CODE RISK ESTIMATION WORKSHEET: OpNav			
ASPECT RISK	ASPECT / Focus		MITIGATION
2.6	STANDARDS /		
	1.6	Stylistic	High level of adherence to a chosen internal or published stylistic coding standard
	3.5	Criticality	High level of adherence to critical-system coding guidelines (MISRA, CERT, etc)

- The OpNav code base shows consistent use of a coding standard.
  - The code's layout and style is uniformly good. Comments indicate the code adheres to NASA's Flight Software Branch C Coding Standard Version 1 cFE Flight Software Application Developers Guide, however SEI did not obtain a copy of this coding standard for review.
  - The code contains numerous violations of typical mission-critical or safety-critical coding standards such as the NASA / JPL Power-of-Ten, MISRA C, or JSF++. Adoption and adherence to a critical-system coding standard is highly recommended by SEI for critical embedded systems like OpNav.

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Standards

Detail

File: \opnav\cfs\apps\ci\fsw\src\ci_custom.c

```
111  int32 CI_CustomInit(void)
112  {
113      int32 iStatus = CI_ERROR;
114      uint32 taskId = 0;
115      IO_TransUdpConfig_t config;
116
117      /* Set Config parameters */
118      CFE_PSP_MemSet((void *) &config, 0x0, sizeof(IO_TransUdpConfig_t));
119      strncpy(config.cAddr, CI_CUSTOM_UDP_ADDR, 16);
120      config.usPort = CI_CUSTOM_UDP_PORT;
121      config.timeoutRcv = CI_CUSTOM_UDP_TIMEOUT;
122
123      if (IO_TransUdpInit(&config, &g_CI_CustomData.udp) < 0)
124      {
125          goto end_of_function;
126      }
127
128      iStatus = CFE_ES_CreateChildTask(&taskId,
129                                       "CI Custom Main Task",
130                                       CI_CustomMain,
131                                       CI_CUSTOM_TASK_STACK_PTR,
132                                       CI_CUSTOM_TASK_STACK_SIZE,
133                                       CI_CUSTOM_TASK_PRIO,
134                                       0);
135  end_of_function:
136      return (iStatus);
137  }
```

- The OpNav code has consistent code style and layout.
- However, OpNav contains many instances of using goto, which is discouraged by most critical-system coding standards.

File: \opnav\cfs\osal\ut_assert\src\utlist.c

```
41  UtListHead_t *UtList_Create(void)
42  {
43      UtListHead_t *NewList;
44
45      NewList = malloc(sizeof(UtListHead_t));
46      NewList->First = NULL;
47      NewList->Last = NULL;
48      NewList->NumberOfEntries = 0;
49      return (NewList);
50  }
```

- The code violates a number of other critical system coding issues such as 43 instances of dynamic memory allocation.
- Some other critical system issues include:
  - Not checking or using non-void return values
  - Low density of assert usage
  - Multi-level pointer dereferencing

Maintainability - Comments

CODE RISK ESTIMATION WORKSHEET: OpNav			
ASPECT RISK	ASPECT /	Focus	MITIGATION
2.7	MAINTAINABILITY /		
Comments	1.2	Quantity	High level of commenting excluding boilerplates
	1.9	Quality	High quality of commenting that clarifies the code's intent

- The OpNav code commenting is excellent.
  - Comment density is appropriate. Module and function headers are used consistently and in-line code comments are used judiciously / as needed.
  - Comment quality is very good. Being written at the level of the code’s intent, they add clarity to the code’s what and why. The comments appear to be accurate and mainly use the older C-style /* */, but some use also the C99 // commenting style, which may be easier to maintain.

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Maintainability - Comments

Detail

File: \opnav\cfs\cfe\fsw\cfe-core\src\es\cfe_es_api.c

```
193  /*
194  ** Function: CFE_ES_ResetCFE
195  **
196  ** Purpose:  Reset the cFE core and all apps.
197  **
198  */
199  int32 CFE_ES_ResetCFE(uint32 ResetType)
200  {
201      int32 ReturnCode;
202
203      if ( ResetType == CFE_PSP_RST_TYPE_PROCESSOR )
204      {
205         /*
206         ** Increment the processor reset count
207         */
208         CFE_ES_ResetDataPtr->ResetVars.ProcessorResetCount++;
209
210         /*
211         ** Before doing a Processor reset, check to see
212         ** if the maximum number has been exceeded
213         */
214         if ( CFE_ES_ResetDataPtr->ResetVars.ProcessorResetCount >
215              CFE_ES_ResetDataPtr->ResetVars.MaxProcessorResetCount )
216         {
217             CFE_ES_WriteToSysLog("POWER ON RESET due to max proc resets (Commanded).\n");
218
219             /*
220             ** Log the reset in the ER Log. The log will be wiped out, but it's good to have
221             ** the entry just in case something fails.
222             */
```

- In-code comments appear to be accurate, are not used too much nor too little, and add clarity and intent to the code.

File: \opnav\cfs\cfe\fsw\cfe-core\src\es\cfe_es_cds.c

```
98   /*****************************************************************
99   /**
100  ** \brief Determines whether a CDS currently exists
101  **
102  ** \par Description
103  **        Reads a set of bytes from the beginning and end of the
104  **        area and determines if a fixed pattern is present, thus
105  **        whether the CDS still likely contains valid data or not
106  **
107  ** \par Assumptions, External Events, and Notes:
108  **          None
109  **
110  ** \return #CFE_SUCCESS          \copydoc CFE_SUCCESS
111  ** \return #CFE_ES_CDS_INVALID   \copydoc CFE_ES_CDS_INVALID
112  ** \return Any of the return values from #CFE_PSP_ReadFromCDS
113  **
114  ******************************************************************
115  int32 CFE_ES_ValidateCDS(void);
```

- API function prototypes utilize doxygen tags to create documentation extracted from the code.
- Overall, the OpNav code is very well-commented, an important benefit for maintenance and sustainment.

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Maintainability - Clarity

CODE RISK ESTIMATION WORKSHEET: OpNav			
ASPECT RISK	ASPECT /	Focus	MITIGATION
2.7	MAINTAINABILITY /		
Clarity	1.2	Consistency	High level of code stylistic consistency (naming, source layout, brace usage, etc)
	2.7	Constants	High level of use of symbolic constants
	1.5	Files / Folders	High level of source file and folder naming consistency and organization
	3.1	Readability	High level of overall code readability
	4.0	Conditionals	Low level of conditional compilation

- The OpNav code clarity is mostly very good.
  - The code shows consistent style, indents, bracing, and module layout.
  - Symbolic constants are well-named, though some embedded literals exist in the code.
  - Files are mostly header / implementation pairs organized in folders per sub-systems.
  - Readability aspects are generally good, though the code's reliance on goto hurts its "structuredness" (Ev(G)) and some high cyclomatic complexity v(G) functions exist.
  - OpNav utilizes many conditional compilation preprocessor directives that detract from the code's clarity. These can also obfuscate the precise source code options / provenance unless compile flag states are available for run-time query.

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Maintainability – Clarity

Detail

File: \opnav\cfs\apps\fm\fsw\src\fm_app.c

```
153    int32 FM_AppInit(void)
154    {
155        char *ErrText = "Initialization error:";
156        int32 Result;
157
158        /* Initialize global data  */
159        CFE_PSP_MemSet(&FM_GlobalData, 0, sizeof(FM_GlobalData_t));
160
161        /* Initialize child task semaphores */
162        FM_GlobalData.ChildSemaphore = FM_CHILD_SEM_INVALID;
163        FM_GlobalData.ChildQueueCountSem = FM_CHILD_SEM_INVALID;
164
165        /* Register for event services */
166        Result = CFE_EVS_Register(NULL, 0, CFE_EVS_BINARY_FILTER);
167
168        if (Result != CFE_SUCCESS)
169        {
170            CFE_EVS_SendEvent(FM_STARTUP_EVENTS_ERR_EID, CFE_EVS_ERROR,
171              "%s register for event services: result = 0x%08X", ErrText, (unsigned int)Result);
172        }
173        else
174        {
175            /* Create Software Bus message pipe */
176            Result = CFE_SB_CreatePipe(&FM_GlobalData.CmdPipe,
177                                       FM_APP_PIPE_DEPTH, FM_APP_PIPE_NAME);
178            if (Result != CFE_SUCCESS)
179            {
180                CFE_EVS_SendEvent(FM_STARTUP_CREAT_PIPE_ERR_EID, CFE_EVS_ERROR,
181                  "%s create SB input pipe: result = 0x%08X", ErrText, (unsigned int)Result);
182            }
183            else
184            {
```

- The code exhibits clean spacing, brace usage, and variable & constant naming differentiation with relatively few embedded magic numbers.

File: \opnav\cfs\apps\opnav\fsw\src\cfsapp\onv_cfsapp_app.c

```
285            goto ONV_CfsApp_AppInit_Exit_Tag;
286        }
287
288        /* Init Executive Task */
289        if(ONV_CfsApp_InitExec() != CFE_SUCCESS)
290        {
291            iStatus = ONV_ERROR;
292            ONV_Common_SendEvent(g_data->HkTlmOut.ucEventMsg,
293                                 ONV_INIT_ERR_EID, CFE_EVS_ERROR,
294                                 "OPNAV - Executive Task Init failed.");
295            goto ONV_CfsApp_AppInit_Exit_Tag;
296        }
297
298    #ifdef _VID_UDP_SEM_
299        int32 StatusOs = CFE_OS_ERROR;
300
301        /* Wait to be sure UDP has created the semaphore */
302        CFE_ES_WaitForStartupSync(OPNAV_WAIT_FOR_STARTUP);
303
304        if(iStatus == CFE_SUCCESS)
305        {
306            /* get UDP handshake semaphore */
307            StatusOs = OS_CountSemGetIdByName(&semId_udp, OPNAV_UDP_SEMID);
308            if(StatusOs != OS_SUCCESS)
309            {
310                ONV_Common_SendEvent(g_data->HkTlmOut.ucEventMsg,
311                                     ONV_INIT_INF_EID, CFE_EVS_ERROR,
312                                     "OPNAV - failed to get udp handshake semaphore: "
313                                     "%d, StatusOs: 0x%x",
314                                     semId_udp, StatusOs);
315                iStatus = ONV_ERROR;
316                goto ONV_CfsApp_AppInit_Exit_Tag;
```

- The code makes heavy use of goto's and pre-processor conditionals which detracts from the otherwise good code clarity.

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Maintainability - Reusability

CODE RISK ESTIMATION WORKSHEET: OpNav			
ASPECT RISK	ASPECT	/ Focus	MITIGATION
2.7	MAINTAINABILITY /		
Reusability	1.5	Independence	High level of machine, OS, database, and compiler independence
	3.8	Utilization	Low level of commented-out or dead code, and unused variables
	4.1	Complication	Low level of complicated implementation constructs
	5.0	Duplication	Low level of duplicated (copy / pasted) code

- The OpNav code reusability could be improved.
  - OpNav has good platform independence.
  - The code has numerous unused variables and functions, with some commented out and no-effect code also present.
  - Control structure nesting is high (max = 14), as are the proportion of functions exceeding HIS limit of 4 for call depth (49.9%) and the use of goto (672 instances).
  - OpNav contains many identical code sections among files in the same or different sub-folders within the folder tree.

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Maintainability – Reusability

Detail

File: \opnav\cfs\cfe\fsw\cfe-core\src\sb\cfe_sb_util.c\

```
547     uint16 CFE_SB_GetChecksum(CFE_SB_MsgPtr_t MsgPtr)
548     {
549     #ifdef MESSAGE_FORMAT_IS_CCSDS
550
551         CFE_SB_CmdHdr_t     *CmdHdrPtr;
552
553         /* if msg type is telemetry or there is no secondary hdr... */
554         if((CCSDS_RD_TYPE(MsgPtr->Hdr) == CCSDS_TLM)||(CCSDS_RD_SHDR(MsgPtr->Hdr) == 0)){
555             return 0;
556         }/* end if */
557
558         /* cast the input pointer to a Cmd Msg pointer */
559         CmdHdrPtr = (CFE_SB_CmdHdr_t *)MsgPtr;
560
561         return CCSDS_RD_CHECKSUM(CmdHdrPtr->Sec);
562
563     #endif
564     }/* end CFE_SB_GetChecksum */
```

- The code contains many (1,404) instances of functions with multiple return points.

- Approximately 22% of OpNav consists of duplicated (copy / pasted) code sections. This makes code harder to change (all instances) and, hence, harder to sustain.

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Fault Tolerance

CODE RISK ESTIMATION WORKSHEET: OpNav			
ASPECT RISK	ASPECT /	Focus	MITIGATION
1.6	FAULT TOLERANCE /		
	2.0	Value Checking	High level of return and parameter value checking
	1.0	Error Handling	Existence of a systematic error handling mechanism
	1.9	Error Logging	Existence of an error / fault logging mechanism
	1.1	Instrumentation	Existence of run-time statistics and timing instrumentation
	2.1	Health Monitoring	Effective use of watchdog and system health monitoring

- OpNav exhibits good fault tolerance.
  - OpNav checks most (but not all) return and parameter values before use.
  - The code uses C-style integer error value checking.
  - Event, error, and exception logs are maintained by OpNav.
  - Runtime processor and timing data is captured by OpNav to performance logs.
  - The code uses a watchdog, though evidence was not found that, prior to a watchdog reset, the health of all tasks / threads is checked.

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Fault Tolerance

Detail

File:\opnav\cfs\psp\fsw\grut699-vxworks6\inc\cfe_psp_config.h

```
50  /*
51  ** Watchdog minimum and maximum values ( in milliseconds )
52  */
53  #define CFE_PSP_WATCHDOG_MIN                  0x00000004U  /*
54  #define CFE_PSP_WATCHDOG_MAX                  0xFFFFFFFEU  /*
55
56  /* 75MHz clk / prescaler = 8 (default in VxWorks SPARC Leon3 BSP)
57   *  = 9,375,000 ticks per second on, 9375 ticks/ms */
58  #define CFE_PSP_WATCHDOG_CTR_TICKS_PER_MILLISEC  9375U
59
```

- OpNav configures and initializes a watchdog timer to reset the system if it is not periodically strobed.
- Code that verifies the health of all threads and tasks prior to strobing the watchdog would be an improvement.

File: \opnav\cfs\tools\utf\src\utf_cfe_psp_memory.c

```
304  int32 CFE_PSP_GetKernelTextSegmentInfo(void *PtrToKernelSegment, uint32 *SizeOfKernelSegment)
305  {
306    int32 return_code;
307    uint32 Address;
308
309    /* Handle Preset Return Code */
310    if (cfe_psp_return_value[CFE_PSP_GETKERNELTEXTSEGMENTINFO_PROC] != UTF_CFE_USE_DEFAULT_RETURN_CODE)
311    {
312      return cfe_psp_return_value[CFE_PSP_GETKERNELTEXTSEGMENTINFO_PROC];
313    }
314
315    if ( SizeOfKernelSegment == NULL )
316    {
317      return_code = CFE_PSP_ERROR;
318    }
319    else
320    {
321      Address = (uint32) (&CFE_ES_RegisterApp);
322      memcpy(PtrToKernelSegment,&Address,sizeof(PtrToKernelSegment));
323      *SizeOfKernelSegment = (uint32) ((uint32) &CCSDS_ComputeCheckSum
324                                       - (uint32) &CFE_ES_RegisterApp);
325      return_code = CFE_PSP_SUCCESS;
326    }
327
328    return(return_code);
329  }
```

- The code skips some return value checks, many on memory copy and set functions, as shown above.
- This example also shows another issue: it uses the sizeof the pointer, rather than that of the segment it points to.

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Security – General

CODE RISK ESTIMATION WORKSHEET: OpNav			
ASPECT RISK	ASPECT /	Focus	MITIGATION
3.5	SECURITY /		
General	3.0	Defensiveness	High level of defensive / secure coding practices
	3.8	Validation	High level of validation of untrusted inputs / interfaces
	2.5	Libraries	Low level of use of potentially unsafe library calls
	3.3	Warnings	Evidence that code compiles with no (or few) warnings
	5.0	Traceability	Effective use of executable and parameter file signatures / anti-tamper measures

- OpNav exhibits good fault tolerance.
  - Defensive programming techniques are utilized, but these can be strengthened.
  - No evidence of encryption or data hashing was identified within OpNav.
  - OpNav employs a modern commercial RTOS, but uses some deprecated library calls.
  - The code supports several build environments, but compilation threw some warnings.
  - As a critical system, consider utilizing hashes and / or digital signatures to verify the executable's boot image and all data sources at start-up / run-time.

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Security – General

Detail

File: \opnav\cfs\apps\io_lib\fsw\src\formats\tctf.c

```
62     #define TCTF_RD_BYPASS_FLG(hdr)          (((hdr).Octet[0] >> 5) & 0x01)
63     #define TCTF_WR_BYPASS_FLG(hdr,val)      ((hdr).Octet[0] = ((hdr).Octet[0] & 0xDF) | \
64                                                (((val) & 0x01) << 5))
65
66     #define TCTF_RD_CTLCMD_FLG(hdr)          (((hdr).Octet[0] >> 4) & 0x01)
67     #define TCTF_WR_CTLCMD_FLG(hdr,val)      ((hdr).Octet[0] = ((hdr).Octet[0] & 0xEF) | \
68                                                (((val) & 0x01) << 4))
69
70     #define TCTF_RD_SCID(hdr)                ((((hdr).Octet[0] & 0x03) << 8) | ((hdr).Octet[1]))
71     #define TCTF_WR_SCID(hdr,val)            ((hdr).Octet[0] = ((hdr).Octet[0] & 0xFC) | \
72                                                (((val) >> 8) & 0x03),     \
73                                               (hdr).Octet[1] = (val))
```

- OpNav contains numerous (281) instances of function-like macros. Inline functions are fast yet more robust, providing compiler parameter type checking.

File: \opnav\cfs\cfe\fsw\cfe-core\src\es\cfe_es_api.c

```
539  int32 CFE_ES_RunLoop(uint32 *RunStatus)
540  {
541      int32  ReturnCode;
542      uint32 AppID;
543      uint32 TaskID;
544
545      CFE_ES_LockSharedData(__func__,__LINE__);
546
547      /*
548      ** Get App ID
549      */
550      ReturnCode = CFE_ES_GetAppIDInternal(&AppID);
551
552      if ( ReturnCode == CFE_SUCCESS )
553      {
554
555         /*
556         ** Get the Task ID for the main task
557         */
558         OS_ConvertToArrayIndex(CFE_ES_Global.AppTable[AppID].TaskInfo.MainTaskId, &TaskID);
559
560         /*
561         ** Increment the execution counter for the main task
562         */
563         CFE_ES_Global.TaskTable[TaskID].ExecutionCounter++;
564
565         /*
566         ** Validate RunStatus
567         */
568         if ( *RunStatus == CFE_ES_RUNSTATUS_APP_RUN || *RunStatus == CFE_ES_RUNSTATUS_APP_E
569         {
```

- OpNav code sometimes uses parameters before verifying them.

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Security – CERT

CODE RISK ESTIMATION WORKSHEET: OpNav			
ASPECT RISK	ASPECT /	Focus	MITIGATION
3.5	SECURITY /		
CERT	1.5	Density	Low density of secure coding issues as per SEI CERT Secure Coding Rules
	2.4	Priority	Low number of high-priority secure coding issues as per SEI CERT Secure Coding Rules

- OpNav exhibits good adherence to most of the CERT Secure Coding rules.
  - A low density of diagnostics were thrown for wide spectrum secure coding rules.
  - Relatively few high-priority secure coding rules showed an appreciable number of diagnostics.

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Testability

CODE RISK ESTIMATION WORKSHEET: OpNav			
ASPECT RISK	ASPECT /	Focus	MITIGATION
1.9	TESTABILITY /		
	1.4	Unit	Evidence of unit testing, especially automated (test cases, test harness, etc)
	1.4	Regression	Evidence of regression testing, especially automated
	2.8	Complexity	Low valued cyclomatic complexity (CC1) and path count

- OpNav exhibits good testability.
  - The code base included extensive unit test code for use with a custom testing framework.
  - Evidence of automated regression testing was identified.
  - Functions within the OpNav code have generally low cyclomatic complexity (CC1) and overall complexity is fairly evenly distributed.

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Testability

Detail-1

File: \opnav\cfs\osal\src\tests\file-api-test\file-api-test.c

```
28      void OS_Application_Startup(void)
29      {
30          errname[0] = 0;
31
32          if (OS_API_Init() != OS_SUCCESS)
33          {
34              UtAssert_Abort("OS_API_Init() failed");
35          }
36
37          /*
38           * Register the test setup and check routines in UT assert
39           *
40           * It is done this way so that the output is logically grouped,
41           * otherwise the entire thing would be lumped together
42           * as a single test case.
43           */
44          UtTest_Add(TestMkfsMount, NULL, NULL, "TestMkfsMount");
45          UtTest_Add(TestCreatRemove, NULL, NULL, "TestCreatRemove");
46          UtTest_Add(TestOpenClose, NULL, NULL, "TestOpenClose");
47          UtTest_Add(TestReadWriteLseek, NULL, NULL, "TestReadWriteLseek");
48          UtTest_Add(TestMkRmDirFreeBytes, NULL, NULL, "TestMkRmDirFreeBytes");
49          UtTest_Add(TestOpenReadCloseDir, NULL, NULL, "TestOpenReadCloseDir");
50          UtTest_Add(TestStat, NULL, NULL, "TestStat");
51          UtTest_Add(TestOpenFileAPI, NULL, NULL, "TestOpenFileAPI");
52          UtTest_Add(TestUnmountRemount, NULL, NULL, "TestUnmountRemount");
53          UtTest_Add(TestRename, NULL, NULL, "TestRename");
54      }
```

- The code makes use of a unit test harness to set up, execute, and tear down test case sets.

File: \opnav\cfs\cfe\fsw\cfe-core\unit-test\evs_UT.c

```
189         /* Test early initialization with an unexpected size returned
190          * by CFE_PSP_GetResetArea
191          */
192         UT_InitData();
193         UT_SetSizeofESResetArea(0);
194         CFE_EVS_EarlyInit();
195         UT_Report(__FILE__, __LINE__,
196                   WriteSysLogRtn.value == EVS_SYSLOG_OFFSET + 2,
197                   "CFE_EVS_EarlyInit",
198                   "Unexpected size returned by CFE_PSP_GetResetArea");
199
200         /* Test task initialization where the application registration fails */
201         UT_InitData();
202         UT_SetRtnCode(&ES_RegisterRtn, -1, 1);
203         CFE_EVS_TaskInit();
204         UT_Report(__FILE__, __LINE__,
205                   WriteSysLogRtn.value == EVS_SYSLOG_OFFSET + 9,
206                   "CFE_EVS_TaskInit",
207                   "Call to CFE_ES_RegisterApp failure");
```

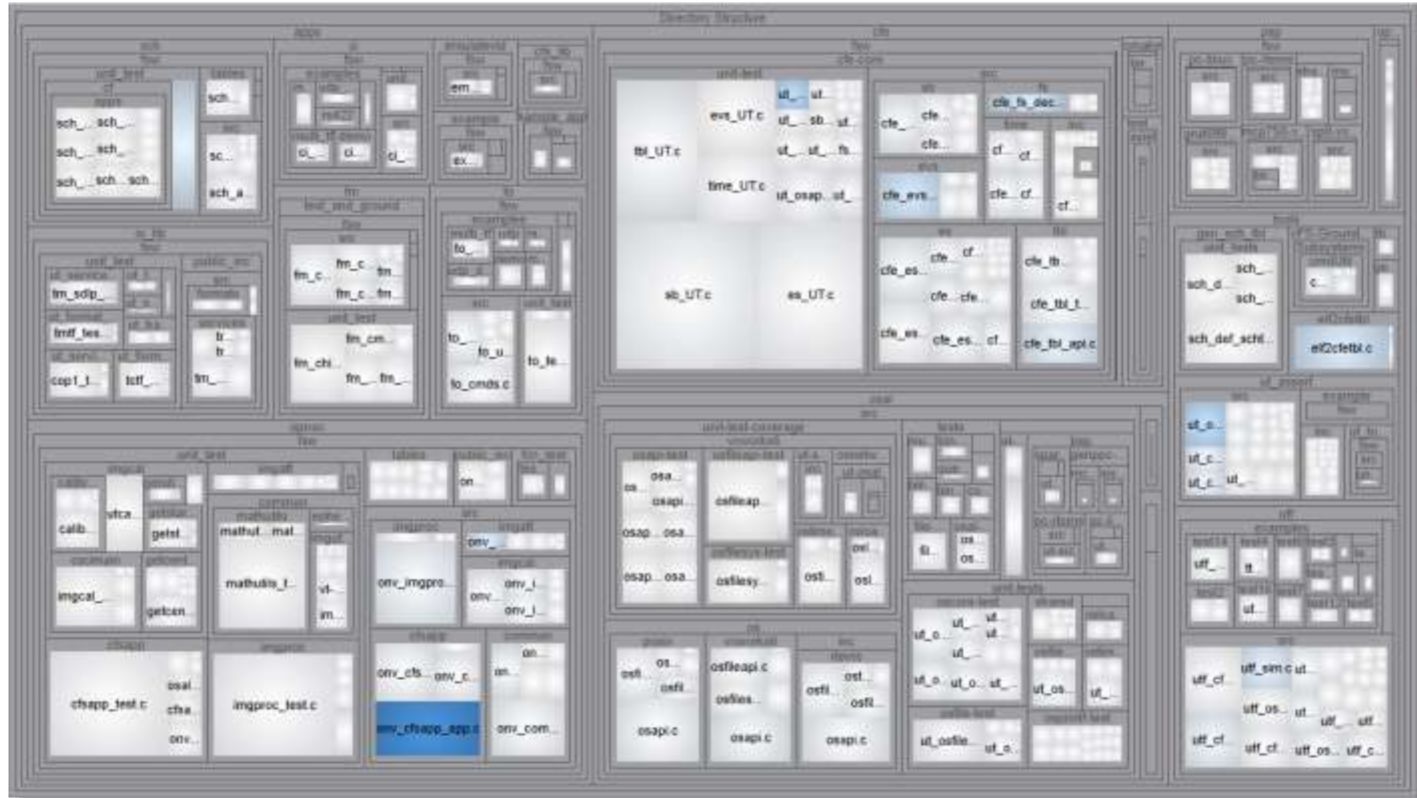
- OpNav unit test code goes beyond happy-paths to test edge cases and failure mechanisms.

Testability

Detail-2

OpNav maximum function cyclomatic complexity (CC1) is generally low and is evenly distributed.

This reduces the difficulty of attaining full-coverage unit testing.



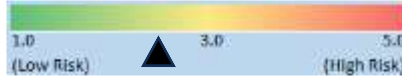
Key:

- Cell Blueness ~CC1
- Cell Size ~SLOC

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Summary

NASA OpNav 1.5 Code Risk Estimation



CREW Total Risk Score = **2.50**

MAINTAIN	IMPROVE
• Good module & code layout consistency and file & folder organization	• Reduce the number of conditional compilation flags in the code and / or make them run-time traceable
• Uniform adherence to stylistic NASA Flight Software Branch C Coding Standard	• Adopt and adhere to a critical coding standard such as MISRA or JSF++ and enforce it via static analysis
• Consistent error handling and logging	• Eliminate dynamic memory allocation after boot-time initialization
• Solid unit testing and (evidence of) automated regression testing	• Eliminate duplicated source files and copy / pasted code sections
• Good run-time code instrumentation & monitoring	• Implement traceability of data & executable files with anti-tamper verification at boot-time
• Excellent accuracy, level of, and clarity of code commenting	• Reduce the number of cyclic and bi-directional dependencies
• Low density of secure coding issues	• Reduce the depth of control structure & call nesting

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.