DEPENDABLE TECHNOLOGIES
FOR CRITICAL SYSTEMS

CRITICAL SOFTWARE, S.A.

HIS ACADEMY TRAINING PLAN

NEURODIVERSITY PROGRAM

CONTRACT REFERENCE: NA

DATE: 2021/09/20
PROJECT CODE: HISACADEMY
DOC. REF.: CSW-2021-DOC-04061
STATUS: DRAFT
PAGES: 8
ACCESS: CONFIDENTIAL PROJECT
VERSION: 01

TABLE OF CONTENTS

1. DEVELOPMENT

1.1. IMPLEMENT SIMPLE C CODE FUNCTIONS

Given a very high level “requirements specification” the trainees shall be able to develop the following simple C code functions:

1.1.1. Mathematical Sum operation

Sum-00: The function shall have the following prototype:

UINT32 sum (UINT32 input_1, UINT32 input_2, BOOLEAN * valid)

Sum-01: The function shall return the result of the mathematical addition operation of first and second input parameters.

Sum-02: The function shall be able to sum values in range [0, 200], otherwise shall signal an invalid operation as per Sum-03.

Sum-03: The function shall set the value referenced by the third parameter to FALSE in case of invalid operation, otherwise shall set it to TRUE.

1.1.2. Mathematical Subtraction operation

Sub-00: The function shall have the following prototype:

UINT32 subtraction (UINT32 input_1, UINT32 input_2, BOOLEAN * valid)

Sub-01: The function shall return the result of the mathematical subtraction operation of first and second parameters (by the specified order).

Sub-02: The function shall be able to subtract values in range [0, 200], otherwise shall signal an invalid operation as per Sub-04.

Sub-03: The functions shall signal invalid operation (as per Sub-04) in case first subtraction parameter is less than the second subtraction parameter.

Sub-04: The function shall set the value referenced by the third parameter to FALSE in case of invalid operation, otherwise shall set it to TRUE.

1.1.3. Mathematical Multiplication operation

Mul-00: The function shall have the following prototype:

UINT32 multiplication (UINT32 input_1, UINT32 input_2, BOOLEAN * valid)

Mul-01: The function shall return the result of the mathematical multiplication operation of first and second input parameters.

Mul-02: The function shall be able to multiply values in range [0, 200], otherwise shall signal an invalid operation as per Mul-03.

Mul-03: The function shall set the value referenced by the third parameter to FALSE in case of invalid operation, otherwise shall set it to TRUE.

1.1.4. Mathematical Division operation

Div-00: The function shall have the following prototype:

UINT32 division (UINT32 input_1, UINT32 input_2, BOOLEAN * valid)

Div-01: The function shall return the result of the mathematical division operation of first and second parameters (by the specified order).

Div-02: The function shall be able to divide values in range [0, 200], otherwise shall signal an invalid operation as per Div-04.

Div-03: The functions shall signal invalid operation (as per Sub-04) in case second subtraction parameter is zero (protect divide by zero).

Div-04: The function shall set the value referenced by the third parameter to FALSE in case of invalid operation, otherwise shall set it to TRUE.

2. TESTING

2.1. PERFORM UNIT TEST ACTIVITIES

The trainees shall develop the unit tests / module tests / LLR tests against the requirements used to developed the C Code functions at sections 1.1.1, 1.1.2, 1.1.3 and 1.1.4.

2.1.1. Objectives

The following objectives are required to be achieved by the developed unit tests:

- Range Test
- Robustness Test
- Discontinuities Test
- 100% Statement Structural coverage
- 100% Branch/Decision Structural coverage
- 100% MCDC Structural coverage

These objectives shall be explained along with the requested test activities.

2.1.2. Concepts

The following main concepts shall be understood to be able to perform unit test activity:

- Discontinuities
- Stubs
- Test inputs
- Test Outputs & checks

These concepts shall be explained along with the requested test activities.

2.1.3. TCF Generator

Please refer to the following presentation that explains how to achieve the objectives at 2.1.1 having the concepts at 2.1.2 in mind.

LDRA is the tool that will be used for this activity. CRITICAL have developed a support excel spreadsheet to build LDRA test case files (.tcf).

High level instructions can be found at:

\HIS-Academy\implementation\90-LDRA-tcf-generator\LDRA911-how-to-run.docx

\HIS-Academy\implementation\90-LDRA-tcf-generator\LDRA TCF Generator_v2.ppt

2.1.4. Test Review

After developing and executing the unit tests, they shall be independently reviewed. The checklist for the review activity is at the following location:

\HIS-Academy\documentation\applicable-documents\02-checklists\verification\LLR_test_spec_review_checklist.xlsm

<u>The following understanding/guidance applies to the checklist questions:</u>

1.1. Are the items under configuration control, follows configuration naming conventions and ready for review?

GUIDELINE: The objective is to ensure Configuration Management activities are correctly followed. Set as N/A because our current configuration environment does not support this feature.

1.2. Have any outstanding failures or observations from previous review activities been addressed?

GUIDELINE: For the first review, set as N/A and justify as "*This is the first review.*". For the subsequent reviews, it must be evaluated accordingly. For instance, if there were no outstanding failures or observations, then it should be set as "Pass" and the following comment may be added: "*There were no outstanding failures or observations from the previous review.*"

1.3. Is the test specification header correct and consistent?

GUIDELINE: For now, set it as N/A. Excel file does not have header. It should be checked as part of .tsp file (not yet available).

2.1. Does a test case exist for each software requirement?

GUIDELINE: The objective of this check is to ensure that all requirements are being covered. In the case of unit testing (a.k.a. module, a.k.a. low level), the objective is to ensure that all low level requirements are covered.

2.2. Is test case correct for each software requirement?

GUIDELINE: Generic check to guarantee that the test cases properly verify the software requirements.

3.1. Have equivalence class ranges (e.g., real and integer variables) been checked for correctness?

GUIDELINE: Checks the nominal values for boundary conditions. For example, if an INT32_T value is valid within [1;200], the mid/min/max should be verified (e.g. 1,100,200). This check is complemented by 4.1.

3.2. For time related functions, such as filters, integrators and delays. multiple iterations of the code should be performed to check the characteristics of the function in context.?

GUIDELINE: It is N/A for low-level testing. It will be checked by SW Integration or HW/SW Integration Testing.

3.3. Have all possible state transitions been verified (e.g. switch statements)?

GUIDELINE: Self-explanatory.

3.4. Have variable usage and boolean operators been verified?

GUIDELINE: Variable usage: checks that there are no overflows in variable value computation. Boolean operators: discontinuities (==,!=,>,>=, …..)

4.1. Have equivalence class ranges (e.g., real and integer variables) been checked **beyond the expected range**?

GUIDELINE: Self-explanatory.

4.2. Have all possible failure modes of the incoming data been determined?

GUIDELINE: Self-explanatory.

4.3. If loop count is a computed value, have out of range loop control values been addressed?

GUIDELINE: Self-explanatory.

4.4. Have invalid state transitions been addressed?

GUIDELINE: Complements 3.3.

4.5. A check should be made to ensure that protection mechanisms for exceeded frame times respond correctly

GUIDELINE: Some functions may implement some kind of "wait" or some scheduling mechanism that should be executed every 10ms and protection must exist to avoid overrun or to properly deal with it.

4.6. Has system initialization been exercised during abnormal conditions?

GUIDELINE: Only applicable to functions that configure HW (e.g register configuration) or implement some kind of SW initialization.

4.7. For time related function, such as filters, integrators and delays, test cases should be developed for arithmetic overflow protection mechanisms

GUIDELINE: This check should be updated to check for all overflows, not only the ones related with "filters, integrators and delays". For now, leave it as N/A because we don't have these constructs.

5.1. Has statement coverage been addressed?

GUIDELINE: Self-explanatory.

5.2. Has decision coverage been addressed?

GUIDELINE: Self-explanatory.

5.3. Has MC/DC coverage been addressed?

GUIDELINE: Self-explanatory.

2.1.4.1. UNIT TEST REVIEW WORKFLOW

WHO	WHAT
Reviewer	Creates LLR Test Specification review checklist. Performs the review, filling in the "Checklist", "RIDs" and "RIDs Summary" worksheets.
Author	Replies to RIDs in column "Author Disposition", accepting or rejecting the RID. If accepts, identifies what shall be done. If rejects, justifies why he thinks it is not a failure. For example, the Author may suggest changing a failure to an observation.
Reviewer	Accepts or rejects the "Author Dispositions", filling in the column "Disposition Acceptance".
Author/Reviewer	Further discussions occur in column "Disposition Acceptance" until a decision is made. If more than one interaction is needed, then a meeting should be considered to clarify the pending issues.
Author	Updates the item under review and submits to review a new revision.
Reviewer	Confirms that the new revision corrects the RIDs. Updates the checklist accordingly.

CRITICAL
SOFTWARE

DEPENDABLE TECHNOLOGIES
FOR CRITICAL SYSTEMS

www.criticalsoftware.com

PORTUGAL | UK | GERMANY | USA

CMMIDEV/5 SM
Exp. 2019-03-24 / Appraisal #25977