QRS-2016
1-3 August 2016
Vienna Austria

Software Reliability Engineering Practice in Embedded RTOS Development

Han Wei , Ye Hong , Mou Ming , Li Yunxi

Aeronautics Computing Technique Research Institute，China

Aug, 2, 2016

Who We Are?

AVIC ACTRI

- AVIC: Aviation Industry Corporation of China
- Founded in 2008;
- Business: General Aviation and Defense, and Avionics Systems;
- 2016, Ranked 143th（Fortune Top 500 Enterprises）
- Over 140 subsidiaries, and more than 400K employees;

- **ACTRI：Aeronautics Computing Technique Research Institute**
- A member of AVIC;

Who We Are?

AVIC ACTRI

ACTRI:

- Located in Xi’an, Shaanxi, P.R. China
- Dedicated Airborne Computer, including:
  - ✓ Airborne Computer For :
    - ✓ Avionics Equipment
      - ✓ Federal Architecture
      - ✓ IMA（ICP etc）
    - ✓ Flight Control
    - ✓ Utility
  - ✓ ASIC/SoC Design;
  - ✓ **RTOS;**
  - ✓ Application SW
- Founded in 1958,
- 1500 employees,
- Income about 3B RMB/Year;

Main Topic

1. Definition and Basis
2. The Software Engineerring of AcoreOS RTOS
3. ACoreOS RTOS Software Reliability Engineering Practice
4. Discussion

1. Definition and Basis

*

What is Embedded RTOS?

- RTOS: Managing the hardware resources, Supply service to hosting applications with very **precise timing and a high degree of reliability**
- RTOS: **“glue” between the middleware**，**application, hardware resources, system services, and input/output (I/O) devices**[2].
- **Widely used** in embedded system industry as Aerospace, **Com/Mil Aircraft**，Automobile, Nuclear, Medical Industries etc.


[1] Technique White Paper-What is a Real-Time Operating System(RTOS) - National Instruments Corporation 2013.11
[2] Study of Commercial Off-The-Shelf (COTS) Real-Time Operating Systems (RTOS) in Aviation Applications – FAA 2002.12

What is Embedded RTOS?

<u>Reliability/Safety</u>

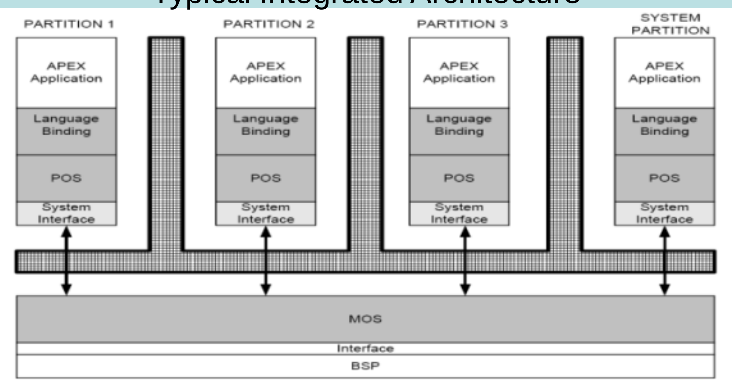
Embedded real-time operating systems and related software tools ***help secure aerospace and defense platforms and mission-critical data from growing threats.***

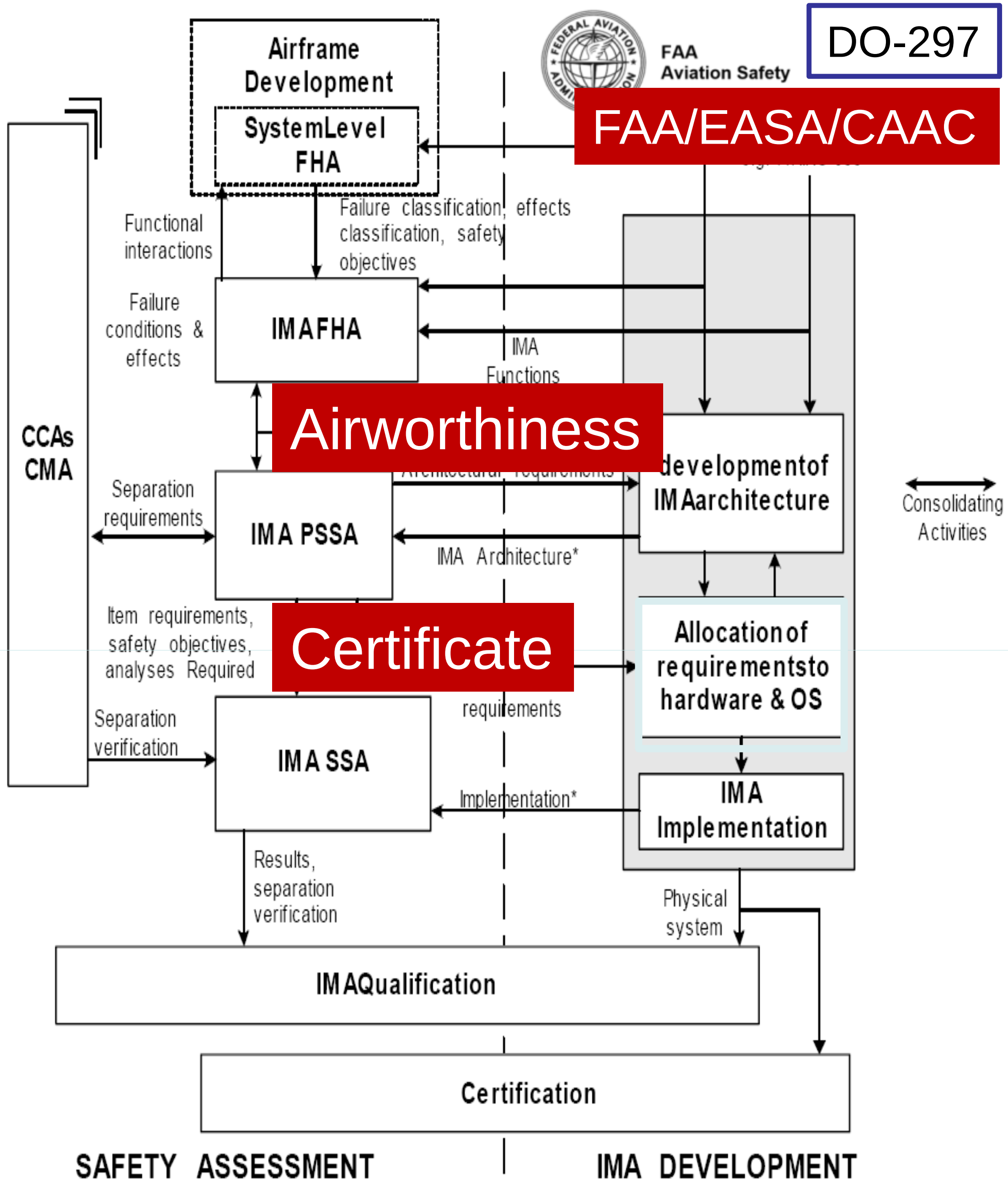
- Courtney E. Howard

REL/SAFETY of RTOS

- RTOS **failure** maybe lead to **system crash** directly;
- **RTOS is key(like determinism) to system safety in application software levels**[1].
- Plays **important** role in IMA system **integration**[2].
- Needs to be developed and verified at the **DAL** of safety associated with the system applications[1].

Typical Integrated Architecture





[1] Study of Commercial Off-The-Shelf (COTS) Real-Time Operating Systems (RTOS) in Aviation Applications – FAA 2002.12
[2] DO-297 IMA System Development Guidance – FAA

Software Reliability: Difficulty things.

- Software Reliability is often very closely tied to **safety critical systems.** And these systems are likely defined as following:
  - ✓ Commercial aircraft (<**10^{-9}**) – flight control - A/C safety
  - ✓ Military Aircraft (<**10^{-7}**)- flight control - A/C safety
  - ✓ Medical (<**10^{-7}**) – Operating control - Patient safety
  - ✓ Automotive (<**10^{-7}**) – passenger safety
  - ✓ Military fire control systems (<$10^{-?}$) – Mission Safety

How to reach the REL Require-ment?

- Complex software reliability is hard to validate, because **the complexity of software.**
  - ✓ Human Development Process: Software Engineering "Try to control"
  - ✓ Functions : complexity
  - ✓ Execution Processes : "Sequence Compose"

How to validate?

Software Reliability Engineering（SRE）

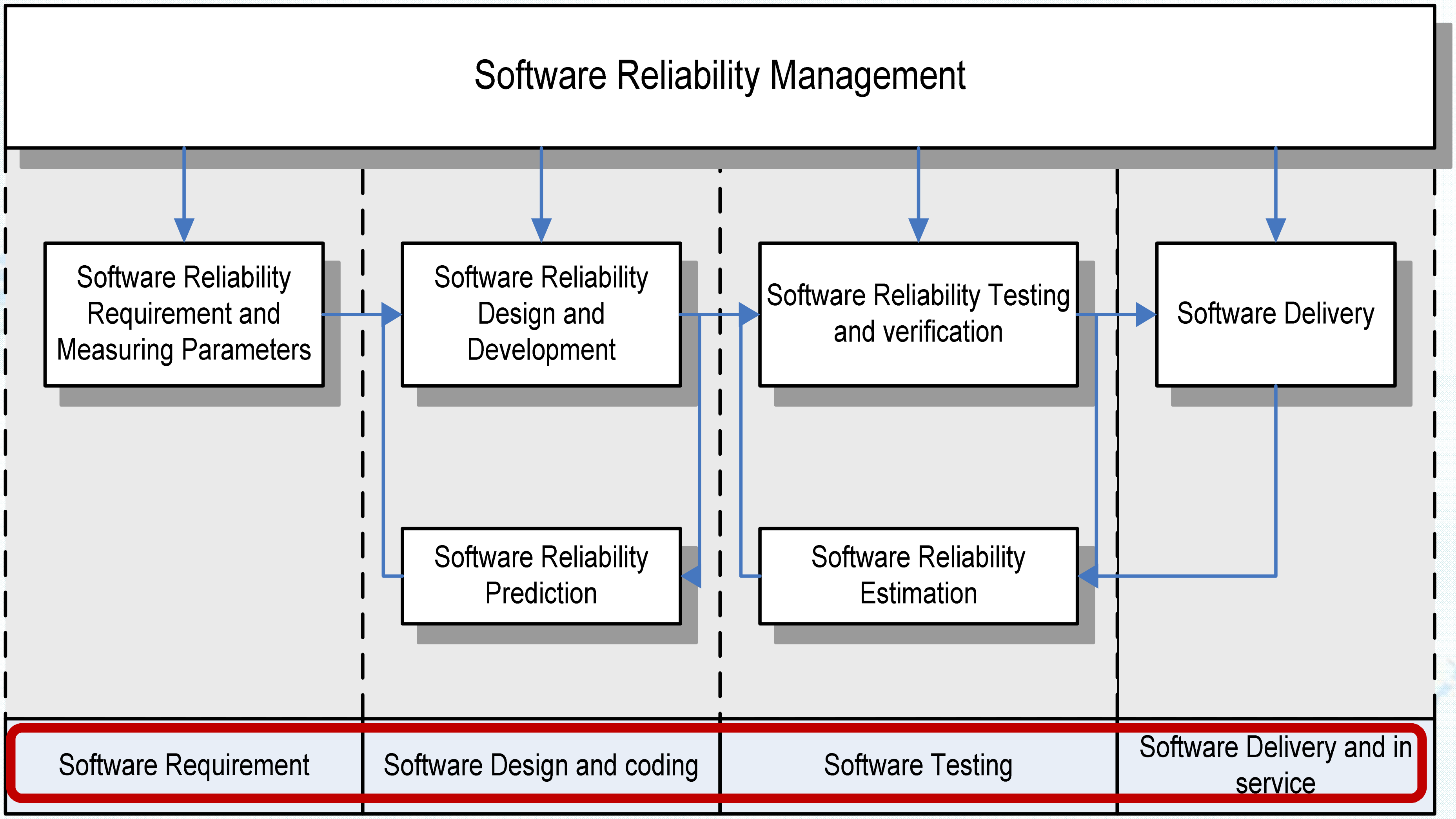
- Problems to be answered in Software Reliability Engineering
  - ✓ 1、How to **development** reliability software?
  - ✓ 2、How the **predict or estimate** software's reliability?

- **1、SRE = Require + Design+ Construction + Verification + Analysis + Management. And** SRE is a **series of** above **activities(or combined) performed to meet** the **S**oftware **R**eliability（**SR**） **Target.**

- **2、Assess of Reliability = Application of statistical techniques to data** collected during system development and operation, Using one of **SR Model** to specify, predict, estimate or assess the reliability.

- The goal of SRE：
  - ✓ Exploring ways of **implementing** "reliability" in software products.
  - ✓ Testing such models and techniques for adequacy, soundness and completeness.
  - ✓ Developing software reliability models and techniques to improve software reliability For SR Growing.

Software Reliability Engineering: Management



2. The Software Engineerring of AcoreOS RTOS

*

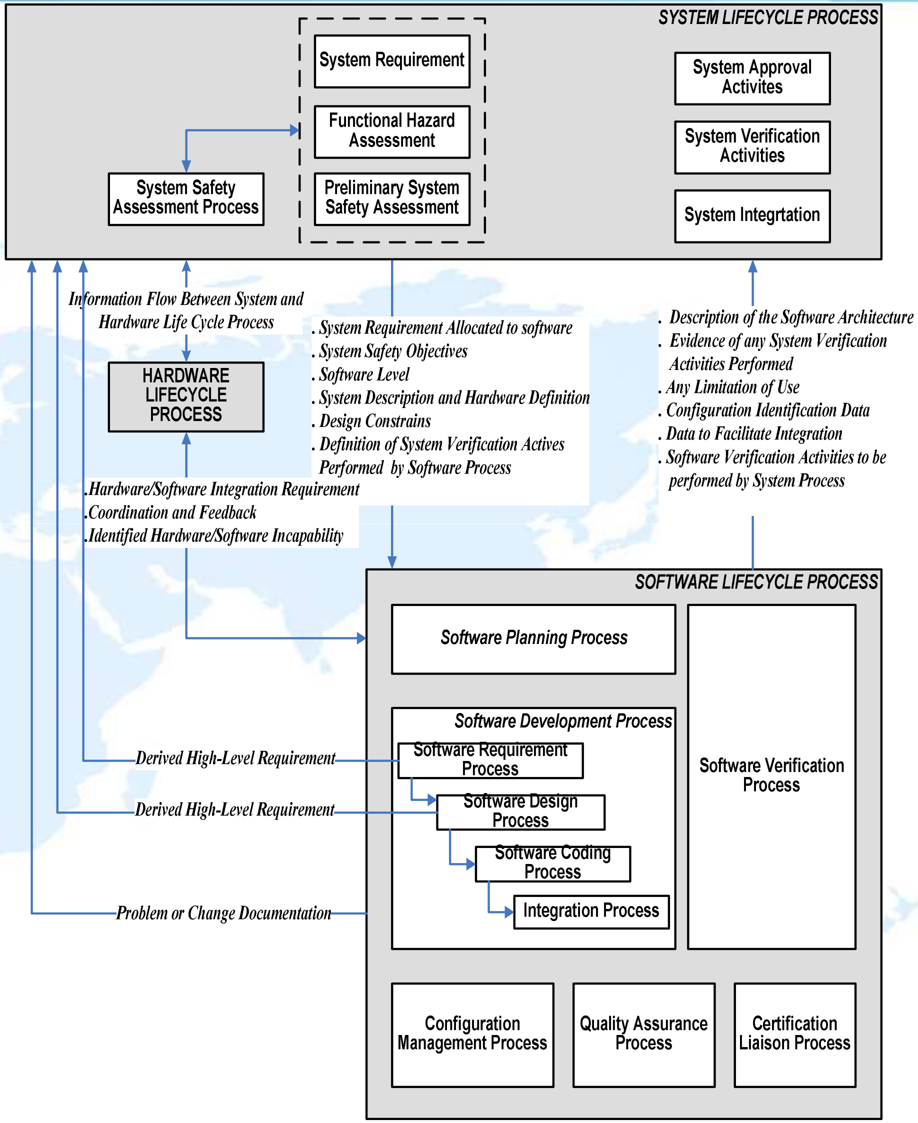
ACoreOS[1] RTOS Family

	ACoreOS1	ACoreOS2
Main Function	Tasks Management, Inter-task Communication and synchronization, Timers, memory management, Device I/O, etc. **FOR Federated avionics**	Partition Management, Inter-partition Communication, Health Monitor, Memory Management etc. **For IMA avionics**
Development Language	C /Assembly	C /Assembly
IDEnvironment	Lambda AE	Lambda AE
Target Hardware	X86, Power, FT(ARM) LS(MIPS),etc	X86, Power, FT(ARM) LS(MIPS),etc
Software Size	100K SLOC	200K SLOC
Design Assurance Level	Level A	Level A
API	ACoreOS， VxWorks 5.X	ARINC 653

[1] ACoreOS is the abbreviation of Avionics Core Operation System.

System and Software

- ACoreOS is **designed for Airborne Computer**, Aviation industry. So Airworthiness Certification(**Conform to DO-178**) is necessary before installed airborne.
- DO-178： an acceptable means for showing **compliance with the applicable airworthiness** regulations for the software aspects of airborne systems and equipment certification[1].
- The information flow between system and software life cycle process[2]
- **For intending to use critical equipment , ACoreOS is defined as DO-178 level A.**

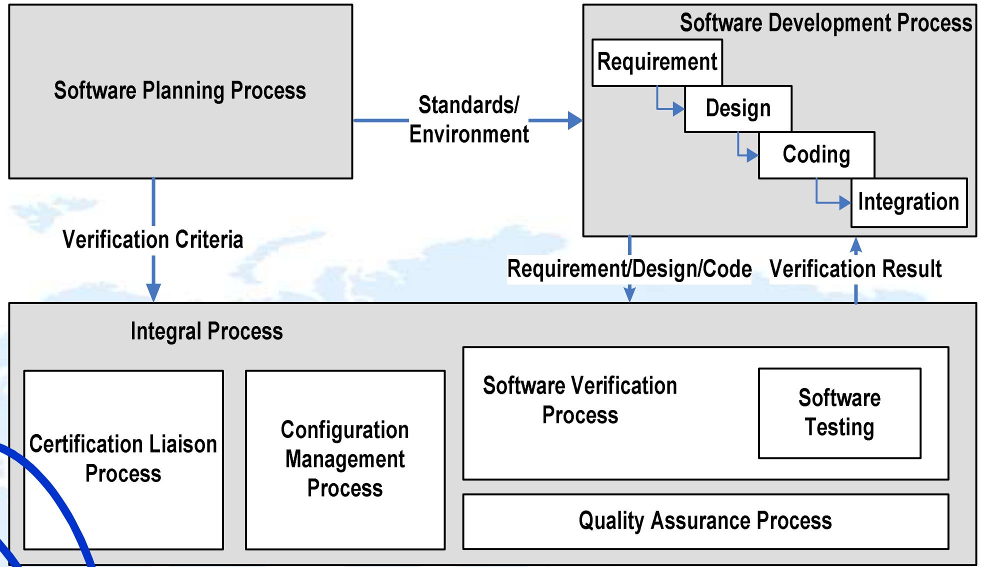


[1] AC-20 115C Airborne Software Assurance – FAA 2013.7
[2] DO-178C Software Considerations in Airborne Systems and Equipment Certification – FAA 2011.11

Level A Software Life Cycle

1. Software Life Cycle is organized as sequenced **processes.**
   - **Software Planning Process**
   - **Software Development Process**
     - ✓ Requirement
     - ✓ Design
     - ✓ Coding
     - ✓ Integration
   - **Software Integral Process**
     - ✓ Verification
     - ✓ Configuration
     - ✓ Software Quality Assurance
     - ✓ Certification Liaison

Activities: defined for each process
Objectives：defined for activities
Evidence：Every activities



DO-178C Process	Objectives
Software Planning Process	7
Software Development Process	7
Verification of Software Requirement process	7
Verification of Software Design Process	13
Verification of Software Coding and Integration	9
Testing of software integration	5
Verification of software verification process	9
Software Configuration process	9
Software Quality Assurance Process	5
Certification Liaison Process	3
Total	71

1. Software Planning

The software planning process for ACoreOS **following DO-178:**

1. Determine the SW Life Cycle
   - Define **Waterfall model** as life cycle model
   - Define **relationships** between the processes
   - Define their **sequencing**
   - Define **feedback** mechanisms
   - Define **transition criteria** from one process to anothers.
2. Define SW Standards
   - SW Requirement Standard
   - SW Design Standard
   - SW Coding Standard
3. Produce the SW Plans
   - SW Development Plan
   - SW Verification Plan
   - SW Configuration Plan
   - SW Quality Assurance Plan
   - Plan for SW Aspects of Compliance[1]

Life Cycle

Standards

Plans

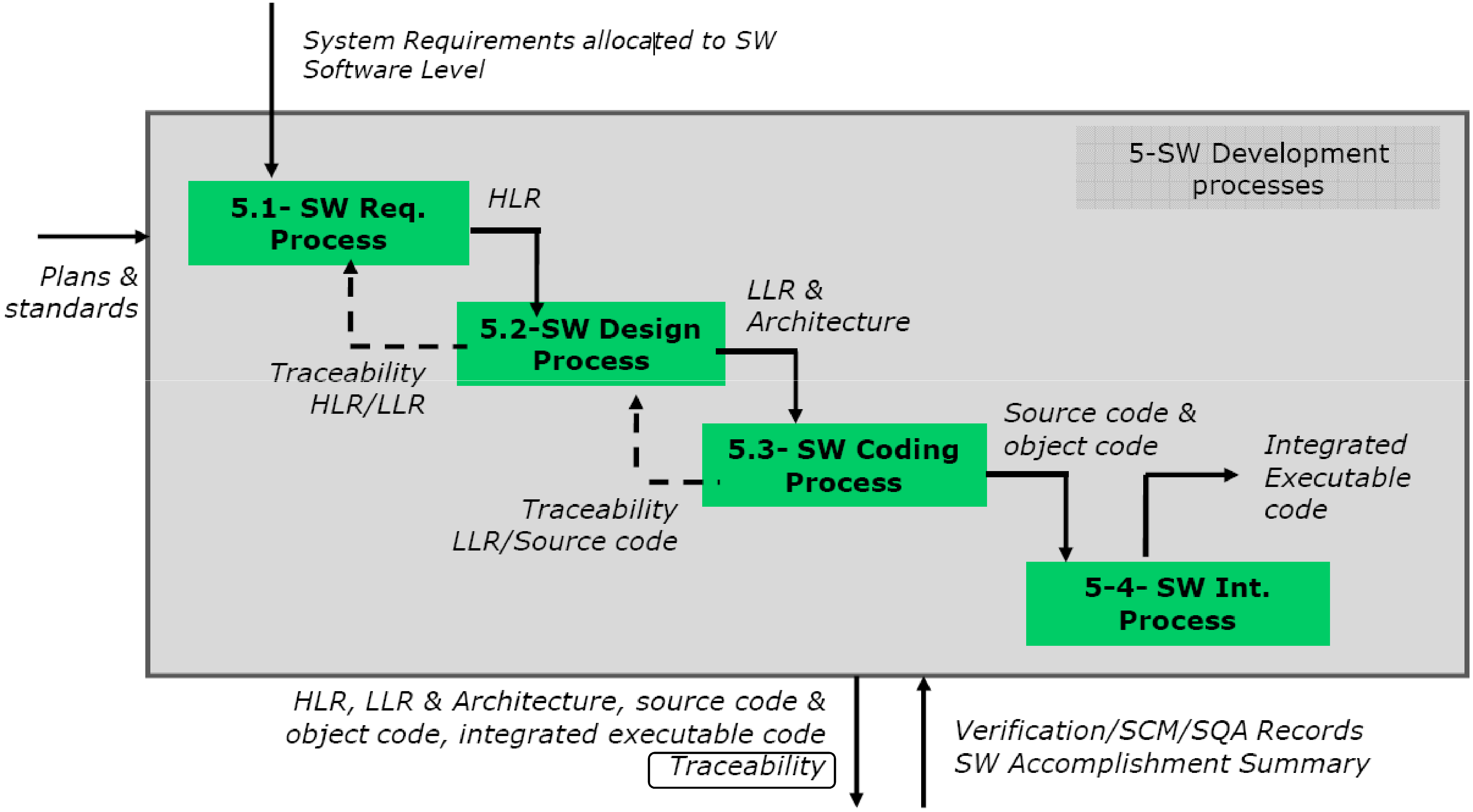
[1] Currently , ACoreOS RTOS has not been certified with any TC of the airborne system /equipment . A DO-178C Compliance inspection is performed. A certification package is supplied to support further TC of airborne system /equipment.

1. Software Planning

The software planning process for RTOS following DO-178(cont.)

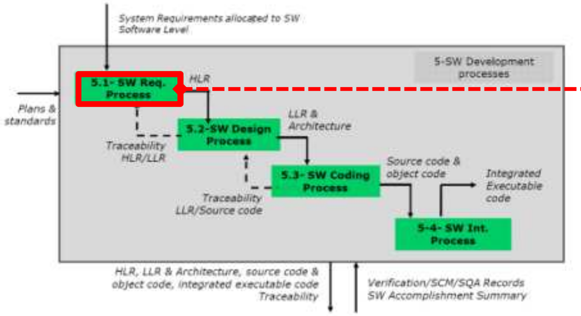
4. Select the SW **Life Cycle environment**（**Tools**）
   - Software **development** environment
     - ✓ Self developed and qualified IDE environment
   - **Project** Management environment ,
     - ✓ DOORs, Reqtify, Synergy, Change
   - **Language** and compiler consideration
     - ✓ C,C++,GNU 2.96, GNU 3.44
   - Software **test** environment
     - ✓ LDRA Testbed
   - ✓ **Target hardware** for RTOS Integration: Intel X86, PowerPC 6XX, PowerPC 7XX, PowerPC 74XX, PowerPC864X(Difficulty to **Qualify the HW platform for RTOS**)
5. Address the additional considerations
   - **Qualification** of IDE environment(Including **Compiler GCC3.4.4**)
6. Plans Are Controlled dynamically
   - **Coding standard** changed from ACoreOS1 to ACoreOS2 period of the project;
   - Revision of the software plans were coordinated within the team leader.

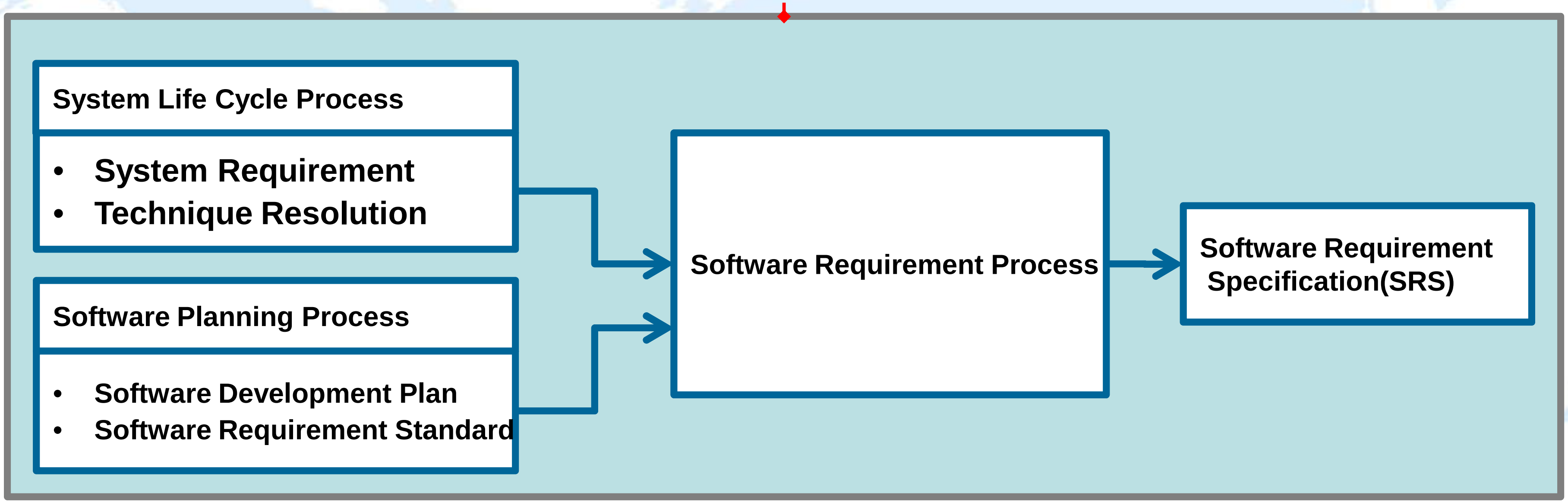
2. Software Development Process



2.1 Software Requirement Process

The Input and output of Software Requirement Process





2.1 Software Requirement Process

Requirement Analysis and SRS

High Level Requirement (HLR)

1. HLR Include:
- Functional Requirement**(API)**
- **Operational** Requirement
- **Constrain and limitation** in scheduling, timing and memory access
- **Interface** between HW and SW
- **System requirement** assigned to software, Health Monitor, Exception Management

2. **444 items are defined for ACoreOS1**
3. **717 items are defined for ACoreOS2**

Low Level Requirement (LLR)

1. LLR : Refinement of HLR, which include:
- **Input/output**
- **Data Structure**
- **Data flow and control flow**
- **Scheduling and communication design**
- **Software Components design**
- **Limitation in resource**

2. **5274 LLRs are defined for ACoreOS1**
3. **3370 LLRs are defined for ACoreOS2.**

Correctness Of SRS:

Unambiguous

Complete,

Verifiable,

Consistent,

Modifiable

Traceable

Robustness

2.1 Software Requirement Process

Requirement Example

RO.1.13.1 Workspace_Allocate

从内核工作空间申请内存块。如果内存块从内核工作空间申请成功，返回内存块的起始地址，否则返回NULL。

派生需求:是

验证方法:分析/测试

全局数据:

Heap_Control _Workspace_Area

Meaning: 管理核心堆的控制块。

Usage: Assignment

Operation Description:

从_Workspace_Area 中申请要求的内存。

功能原型:

```
ACOREOS_INLINE_ROUTINE void
*_Workspace_Allocate(
UINT32 size
)
```

功能位置:

Internal

输入:

UINT32 size /* 请求的内存大小，其单位为字节 */

输出:

None

返回值:

void*

错误:

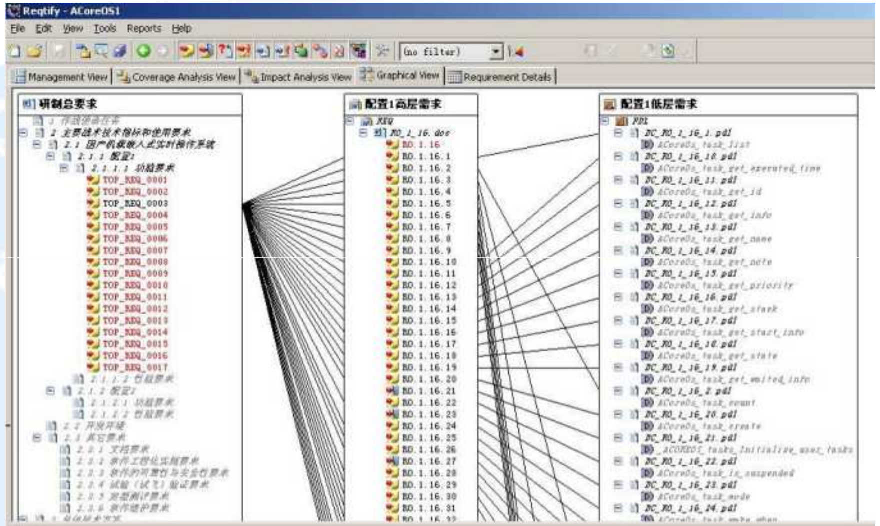
None

注释:

这是一个内联函数。

通过RO.1.3.15中的<_Heap_Allocate >申请内存。

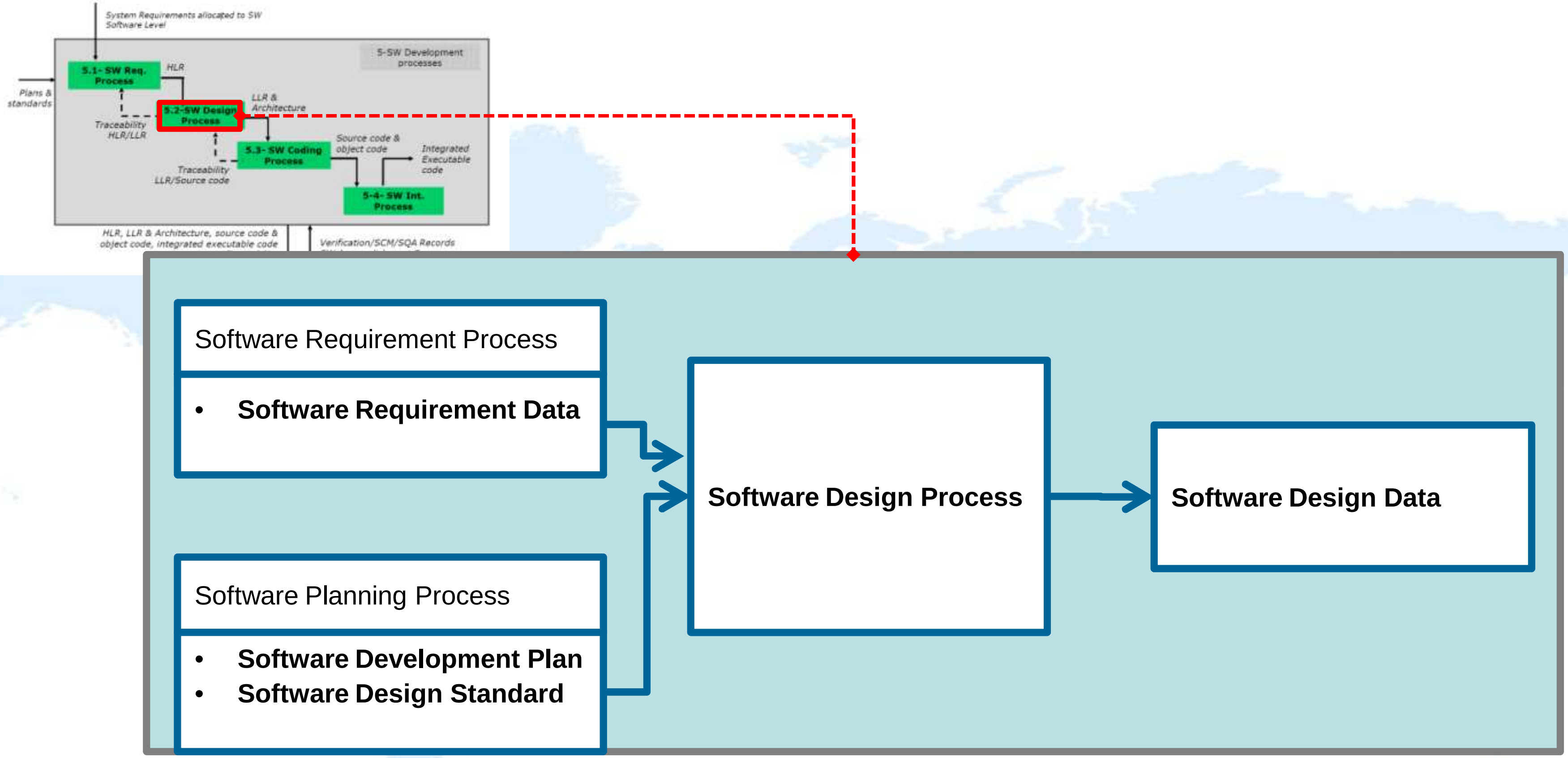
1. Requirements are development following Requirement **Standards**
2. The **traceability** is maintained via **tools** （DOORs and Reqtify）。



1. **Derived requirements** are defined and analyzed to ensure that the high level requirements are not compromised;
2. Derived requirement are **feedbacked** to the system safety Assessment process;

2.2 Software Design Process

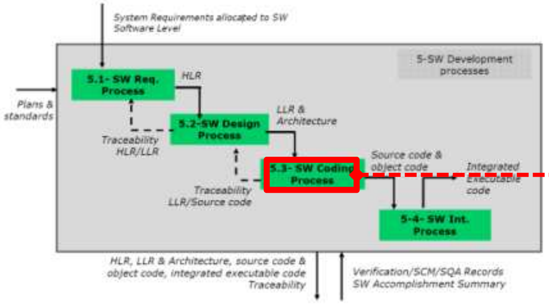
The Input and output of Software Design Process

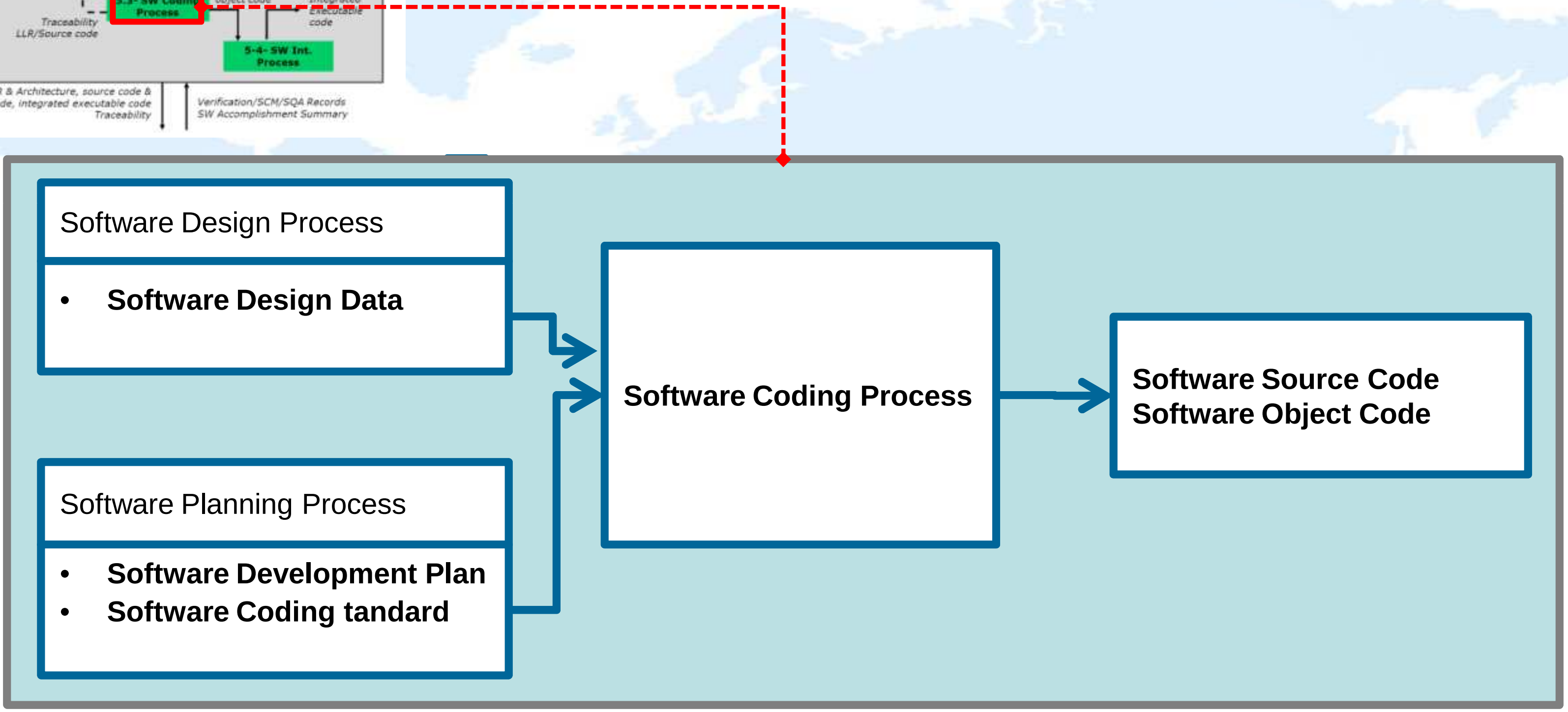


- Design(Or LLR) and software architecture conform to the software design standard and be traceable and consistent;
- LLR are traced to HLR via Reqtify and DOORs tools;

2.3 Software Coding Process

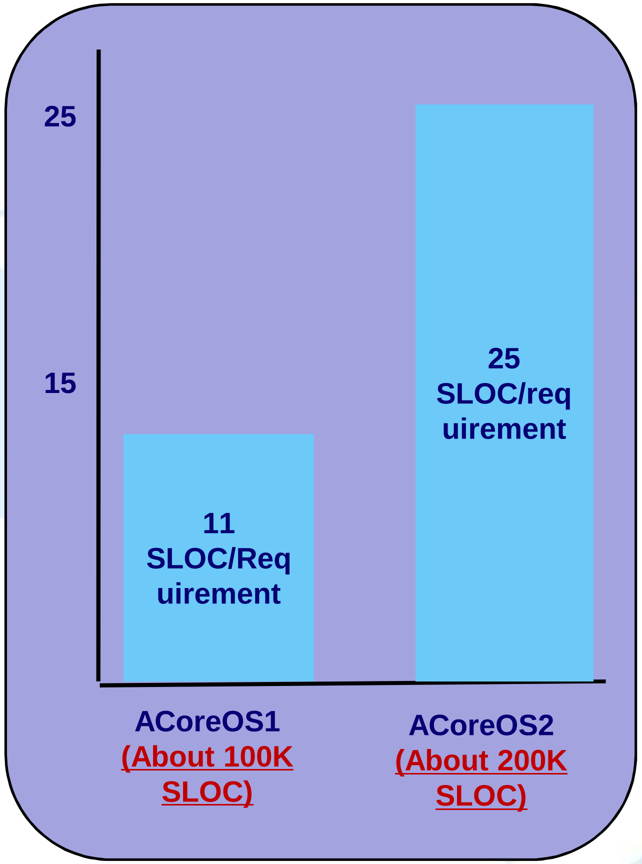
The Input and output of Software Coding Process





2.3 Software Coding Process

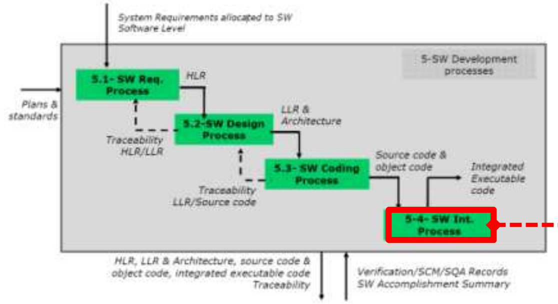
- The Source Code implement the design（LLR）and **conform to the software architecture**;
- The Source Code **conform to the CODE Standards**;
- The Source Code is **traceable** to the Software Design Description via DOORs or Reqtify.
- If inadequate or incorrect inputs is detected during the software coding process，it should **feedback and provided** to the software requirements process, software design process or software planning process as for **clarification or correction**;

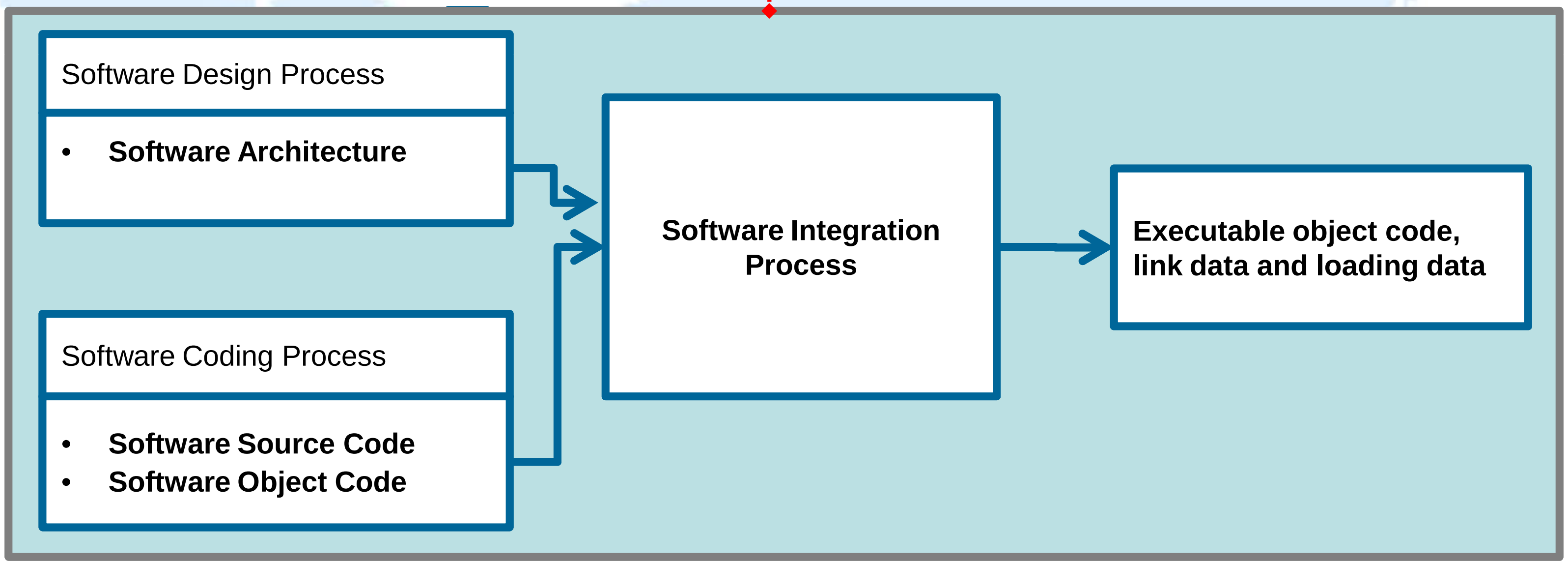


- **ACoreOS1 : About 11 SLOC/per requirement;**
- **ACoreOS2 : About 25 SLOC/per requirement;**

2.4 Software Integration Process

The Input and output of Software Integration Process

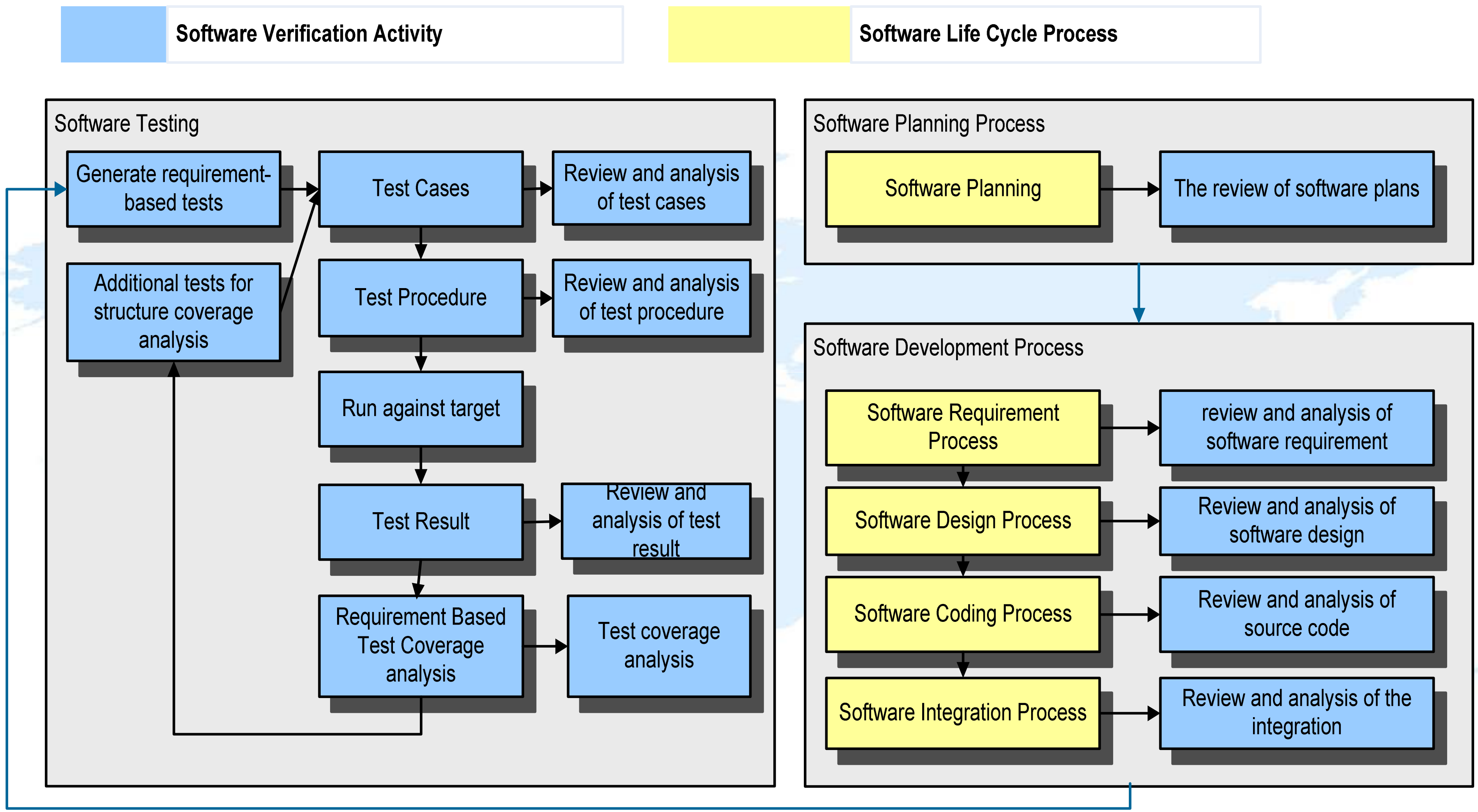




2.4 Software Integration Process

1. Software Integration Process include:
   - **SW/SW integration**
     - ✓ Link/Generate executable code using qualified Compiler
   - **SW/HW integration** using **Qualified HW platform**
     - ✓ Load the executable code in the target computer
2. Deactive Code and Dead Code are processed during integration
   - **The deactivated code** is disabled for the special environment where it is not intended to use. The code to support the running of one type of equipment could be deactive for another kind of equipment. The should demonstrate the disable condition(like using On-Ground-Switch OR C919/C929-Switch) for CAAC.
   - **Dead codes** is unrunning for any environment and should removed out totally.

3. Software Verification Process



40 Verification out of total 66 Targets （DO-178B）

3. Software Verification Process

中航工业

ACoreOS 中航天脉

Embedded Operating System

3.1 Software testing

- Doing according to Level A software;
- Requirement-Based Test.
- **Focus Robustness Test**

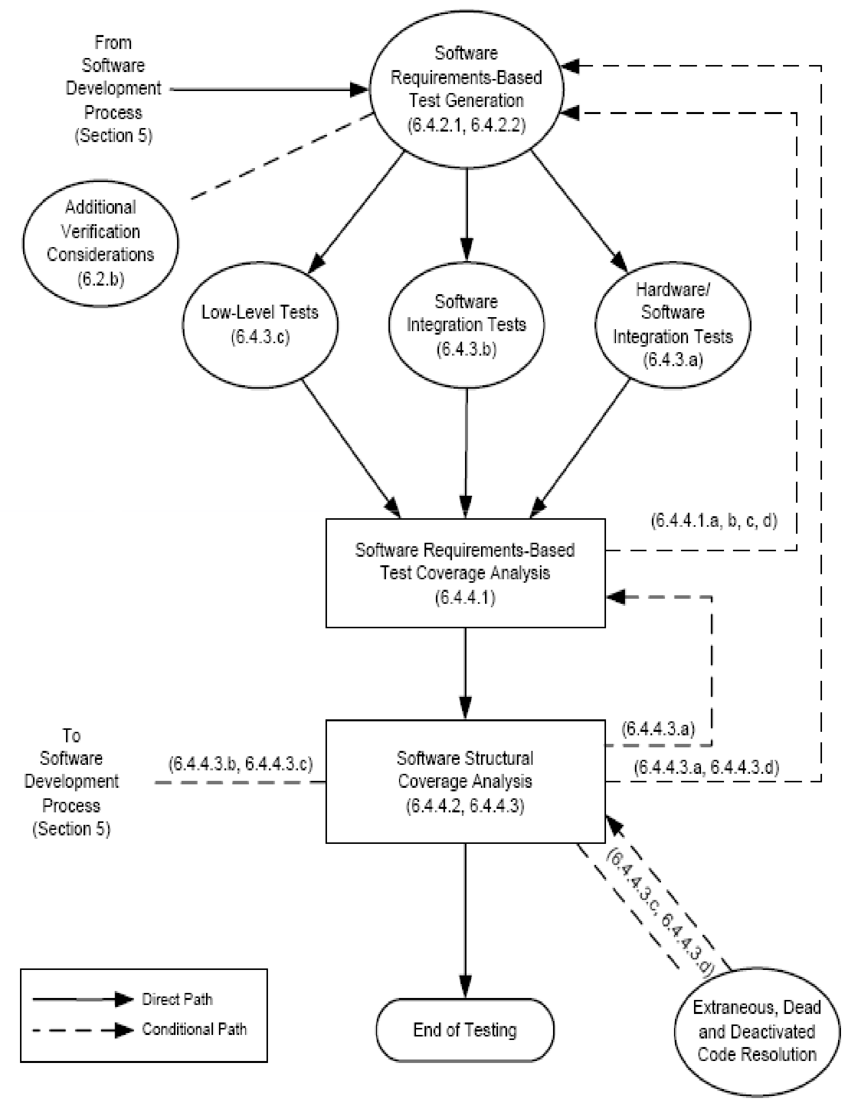
11158 test cases were generated for ACoreOS1,
- Normal condition tests: 4686
- **Robustness tests:5913 (53%)**

18156 test cases were generated for ACoreOS2,
- Normal condition tests: 12586
- **Robustness tests:5570 (31%)**

- Verification Coverage after analysys:
  - ✓ HLR/LLR requirement coverage 100%;
  - ✓ Statement coverage 100%
  - ✓ Decision coverage 100%
  - ✓ MC/DC coverage 100%
  - ✓ Object Coverage 100%

  Before analysys about 85% to 90%

A LEVEL DO-178B (To Be Certificated)



Example of Test Coverage Analysis Reports

B33. log1p()

B33.1. SCA 结果报告

Code File Path: …\target\acoreos2x\workspace\math\libm\math\

SCA Results Report Path: …\SCA 报告\log1p_Coverage.mht

B33.2. SCA 结果报告分析

B33.2.1. 语句覆盖

B33.2.1.1. 源代码 265 行未覆盖

源代码 265 行未覆盖，参见下图中右列蓝色“0 *** ”。

```
390 (265)    if                              10        7        17
391            (                             10        7        17
392            x != x                        10        7        17
393            )                             10        7        17
394            {                             0 ***     0 ***    0 ***
395              return                      0 ***     0 ***    0 ***
396              ( x ) ;      /* x   NaN */  0 ***     0 ***    0 ***
397                ;                         -         -        -
```

B33.2.1.1.1. 结论

测试工具限制，未覆盖部分无法通过添加测试用例方式增加覆盖率。

B33.2.1.1.2. 解决方法

经分析，当系统内部出现故障时，此段代码会被覆盖。

1.1.1.1. 结论

源代码 52 -- 72 行，52 -- 81 行， 79 -- 94 行和 88 -- 94 行未覆盖原因：在当前测试环境中，如果测试条件满足源代码 72 和 81 行，测试程序将陷入死循环（参见 RO1.1.25_21jul09_answer.doc）。由于测试环境的限制，代码不可达。

源代码 52 – 94 行未覆盖原因：switch 语句没有 default 分支。

未覆盖行不能通过增加测试用例满足覆盖。

1.1.1.2. 解决方法

源代码 52 – 72 行，52 – 81 行， 79 -- 94 行和 88 -- 94 行，本未覆盖代码已在测试测试程序 tp_RO_01_01_25.c 中第 714 行至 730 行进行了测试。基于测试程序运行无误，证明源代码正确并且与需求一致。通过对测试用例、代码的分析表明本代码将被正确执行并被覆盖。

源代码 52 – 94 行，添加 default 语句。

本未覆盖代码所属程序单元的测试程序已进行了测试。基于测试程序运行无误，证明源代码正确并且与需求一致。通过对测试用例、代码的分析表明本代码将在不可预知的错误发生时被正确执行并被覆盖。

1. RO.1.1.25 _Watchdog_Tickle()

1.1. SCA 结果报告

Code File Path: …\target\acoreos1x\workspace\kernel\core\src\base

SCA Results Report Path: …\RO1.1\SCA\SCA 报告\watchdogtickle.mht

1.2. SCA 结果报告分析

1.2.1. 语句覆盖

源代码 72， 80， 81 和 89 行未覆盖，参见下图中右列蓝色"0 ***"。

```
92 (72)    case WATCHDOG_INACTIVE :                                      0    0 ***   0 ***
93 (74)        /*                                                        -    -       -
94 (75)         * This state indicates that the watchdog is not on any  -    -       -
95 (76)         * Thus, it is NOT on a chain being tickled. This case    -    -       -
96 (77)         * never occur.                                           -    -       -
97 (79)         */                                                       -    -       -
98           break ;                                                     0    0 ***   0 ***
99 (81)    case WATCHDOG_BEING_INSERTED :                                0    0 ***   0 ***
100 (83)       /*                                                        -    -       -
101 (84)        * This state indicates that the watchdog is in the pro  -    -       -
102 (85)        * BEING inserted on the chain. Thus, it can NOT be on    -    -       -
103 (86)        * being tickled. This case should never occur.           -    -       -
104 (88)        */                                                       -    -       -
105          break ;                                                     0    0 ***   0 ***
```

1.2.1.1. 结论

在当前测试环境中，如果测试条件满足源代码 72 和 81 行，测试程序将陷入死循环（参见 RO1.1.25_21jul09_answer.doc）。由于测试环境的限制，代码不可达，未覆盖行不能通过增加测试用例满足覆盖。

1.2.1.2. 解决方法

本未覆盖代码已在测试测试程序 tp_RO_01_01_25.c 中第 714 行至 730 行进行了测试。基于测试程序运行无误，证明源代码正确并且与需求一致。通过对测试用例、代码的分析表明本代码将被正确执行并被覆盖。

1.2.2. 分支覆盖

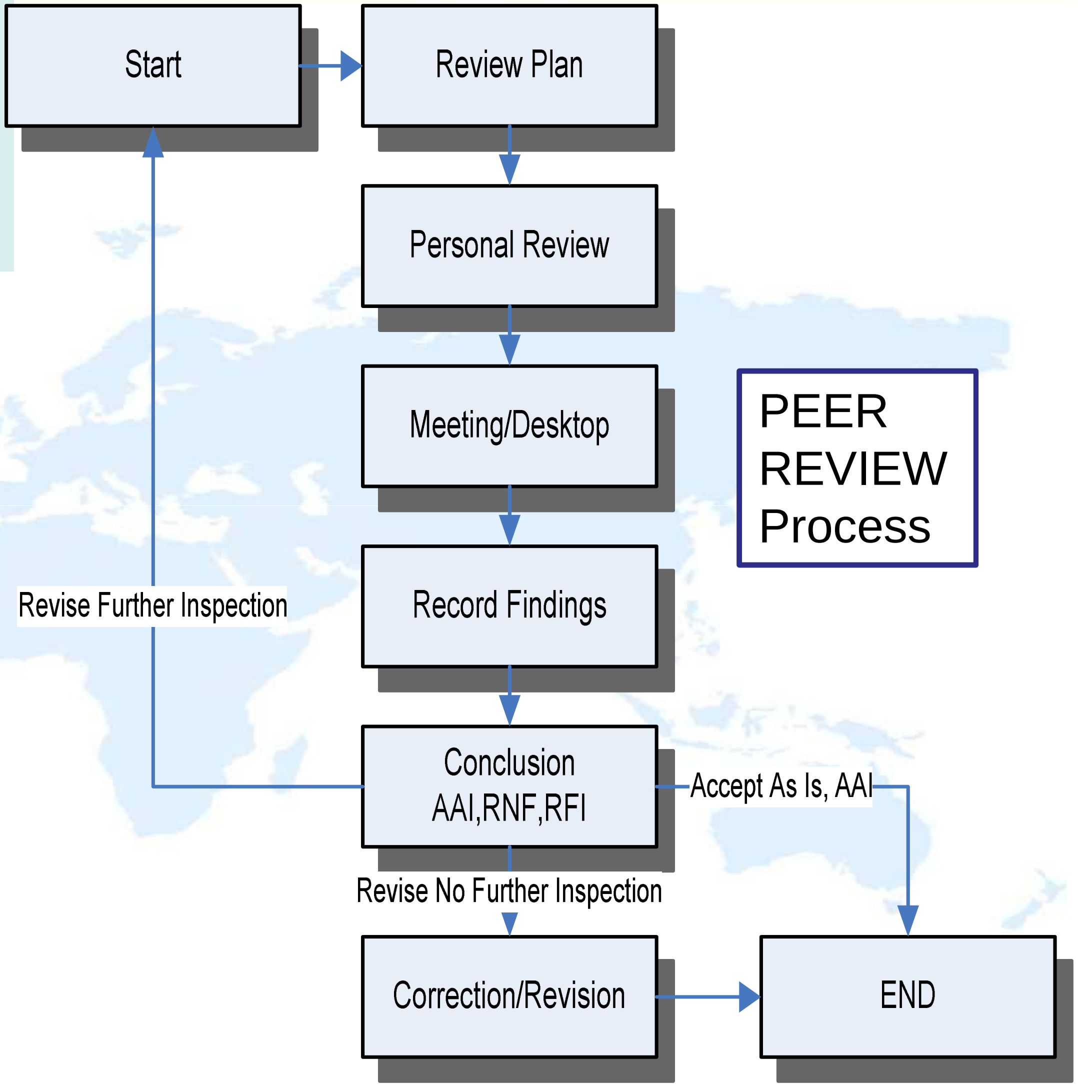
源代码 52 -- 72 行，52 – 81 行，52 -- 94 行，79 -- 94 行和 88 -- 94 行未覆盖，参见下图中右列蓝色"0 ***"。

```
65 (52)    92 (72)     0    0 ***   0 ***
65 (52)    99 (81)     0    0 ***   0 ***
65 (52)    106 (90)    0    2       2
65 (52)    109 (94)    0    0 ***   0 ***

70 (56)    71 (57)     0    3       3       _Is_repetitive ( the_watchdog ) )
70 (56)    90 (65)     0    1       1

75 (58)    76 (59)     0    1       1       _Is_unlimited ( the_watchdog ) )
75 (58)    80 (61)     0    2       2

79 (60)    89 (63)     0    1       1       else

84 (61)    85 (62)     0    1       1       the_watchdog -> repeat_remain -- != 1 )
84 (61)    88 (63)     0    1       1

91 (70)    109 (94)    0    4       4       break ;

98 (79)    109 (94)    0    0 ***   0 ***   break ;

105 (88)   109 (94)    0    0 ***   0 ***   break ;
```

3. Software Verification Process

3.2 Review and Analysis

- Go through ALL Process;
- Review all outputs from each process;
- Strictly follow defined working flow;



ACoreOS1:
Total Peer Review Times: **1286**
Include:
- Software Development Team:**504**
- Software Verification Team:**782**

ACoreOS2:
Total Peer Review Times: **1211**
Include:
- Software Development Team:**647**
- Software Verification Team:**564**

3. ACoreOS RTOS Software Reliability Engineering Practice

*

Software Reliability Procedure

- Software Reliability assess procedure for ACoreOS RTOS software follows that defined in **IEEE 1633-2008(IEEE Recommended Practice on Software Reliability)**.
- A 11-step procedure for assessing and predicting ACoreOS RTOS Software Reliability is executed.

  1. Identify application
  2. Specify the requirement
  3. Allocate the requirement
  4. Define errors(bugs), faults, and failures
  5. Characterize the operational environment(Time/Input)
  6. Select tests
  7. **Select models**
  8. **Collect data**
  9. **Estimate parameters**
  10. **Validate the model**
  11. **Perform assessment and prediction analysis**

- **Exponential Non-Homogeneous Poisson Process(NHPP) models**
  - **Schneidewind model**
  - **Shooman model**
  - **Musa basic model**
  - **Jelinski and Moranda model**
  - **Generalized exponential model**
- **Non-exponential NHPP models**
  - **Duane model**
  - **Brooks and Motley binomial and Poisson model**
  - **Yamada s-shaped model**
  - **Musa and Okumoto logarithmic Poisson model**
- **Bayesian models.**
  - **Littlewood Model**

Candidate Software Reliability Model

- Over 200 models
- Trying to understand the characteristics of software failure(how and why)
- Trying to qualify
- No single model can be used in all situations.
- No model is complete or even representative.

Criteria for conducting an evaluation of Reliability Models in the RTOS:

- **Future predictive accuracy**: Accuracy of the model in making predictions beyond the time span of the collected data.
- Generality: Ability of a model to make accurate predictions in a variety of operational settings (e.g, accurate in different environment).
- **Insensitivity to noise:** The ability of the model to produce accurate results in spite of errors in input data and parameter estimates.

Schneidewind Model

Schneidewind model is selected for follow reason:

- It is the preference model **recommended in IEEE 1633-2008**;
- Comparing to other model, The basic philosophy of Schneidewind model is more **suitable for RTOS software**[1].
- Consider **Defined evaluation criteria above, THIS MODEL is good.**

1. **Requirement oriented** software reliability analysis, Following data are retrieved and collected.

2. **Testing oriented** software reliability parameters analysis.

3. **Trustful analysis** of result based testing.

Requirement oriented software reliability analysis

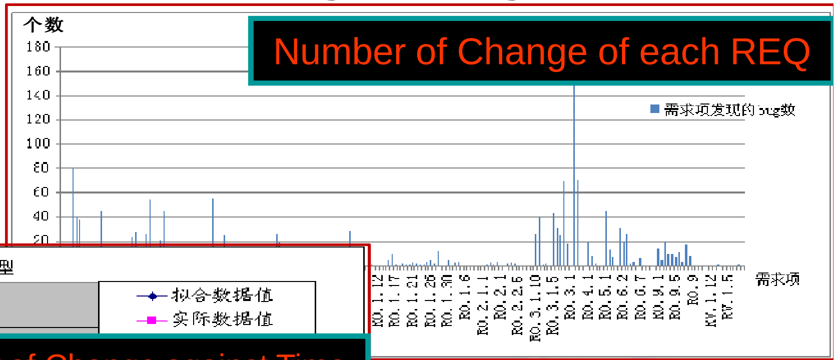
Requirement oriented software reliability analysis

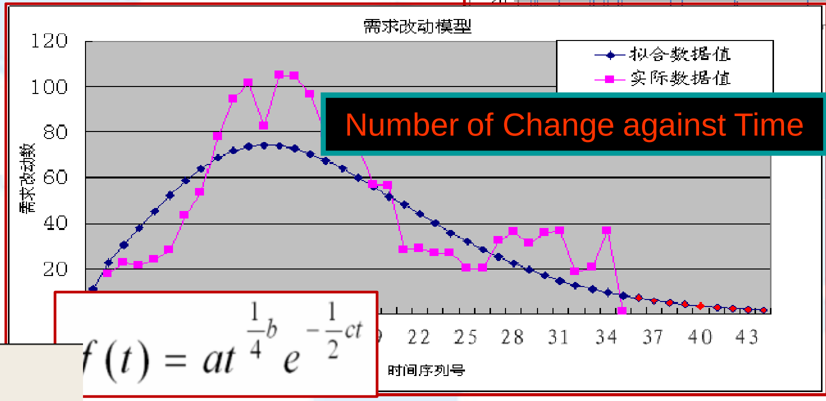
During requirement analysis process, After data catalog/process(smoothly/average/de-odd-point etc.) , We get following big-data:

- **Number of change** of each REQ item from birth to documented.
- **Change times against to time scale**: Number of Changes in ONE time. The Change can be REQ ADD/DEL/CHANGE;
- Number of REQ bugs effect the area against number of REQ items.
- Number of REQ changed, Focus trend of **number of bugs Accumulated**.
- Number bugs of alive, Against time from finding to change correctly.

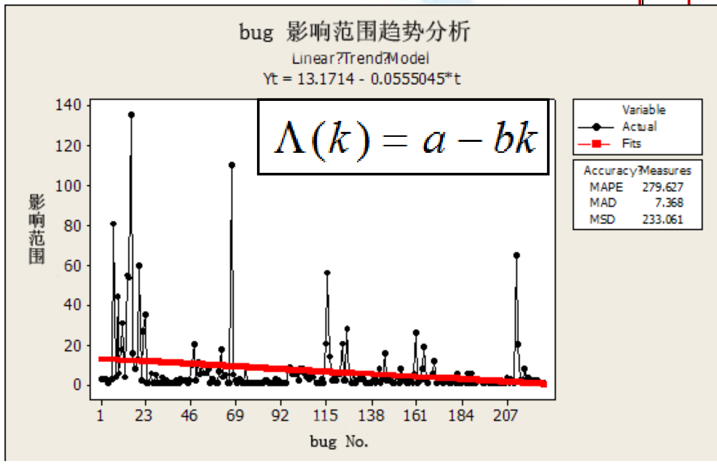
Requirement oriented software reliability analysis

The stability of requirement is a key factor to software reliability!





$$f(t) = at^{\frac{1}{4}b} e^{-\frac{1}{2}ct}$$

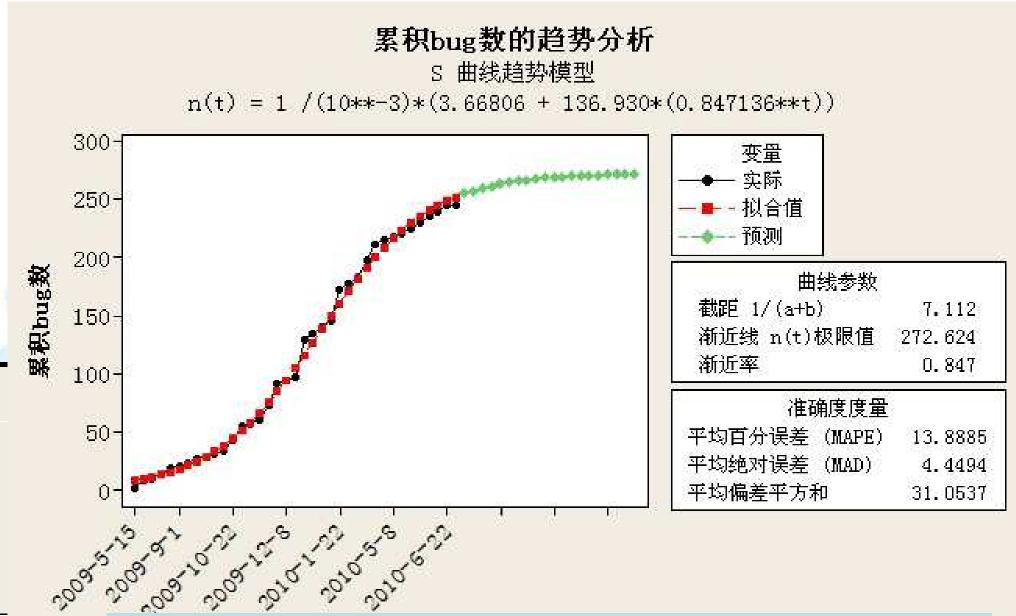


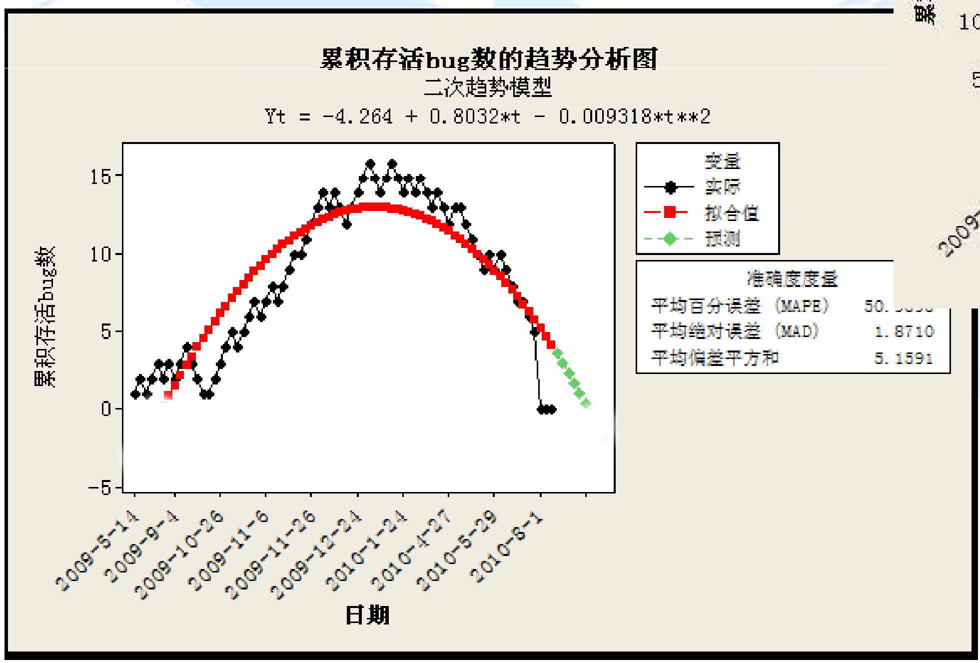
$$\Lambda(k) = a - bk$$

Analysis indicates that the changes of requirements tend to zero.

Requirement oriented software reliability analysis

The stability of requirement is a key factor to software reliability!





To Software Quality, REQ quality is very important.
REQ poor or change frequently are Main reasons for project failure.
This Analysis can give aviation people a confidences.
Dependablity.

Testing oriented software reliability analysis

Schneidewind model

Schneidewind model structure is defined as below:

$$\log L = X_t\left[\log X_t - 1 - \log\left(1 - e^{-\beta t}\right)\right] + X_{s-1}\left[\log\left(1 - e^{-\beta(s-1)}\right)\right] + X_{s,t}\left[\log\left(1 - e^{-\beta}\right)\right] - \beta\sum_{k=0}^{t-s}(s+k-1).$$

The parameters are as follows:

- α Failure rate at the beginning of interval S
- β Negative of derivative of failure rate divided by failure rate (i.e., relative failure rate)
- r_c Critical value of remaining failures; used in computing relative criticality metric (RCM) $r(t_t)$
- S Starting interval for using observed failure data in parameter estimation
- t Cumulative time in the range $[1, t]$; last interval of observed failure data; current interval
- T Prediction time
- t_m Mission duration (end time-start time); used in computing RCM $T_F(t_t)$

The observed quantities are as follows:

$$\lambda(t) = \alpha\exp(-\beta t)$$

- T_{ij} Time since interval i to observe number of failures F_{ij} during interval j; used in computing MSE_T
- X_k Number of observed failures in interval k
- X_i Observed failure count in the range $[1, i]$
- X_{s-1} Observed failure count in the range $[1, s-1]$
- $X_{s,l}$ Observed failure count in the range $[i, s-1]$
- $X_{s,i}$ Observed failure count in the range $[s, i]$
- $X_{s,t}$ Observed failure count in the range $[s, t]$
- $X_{s,t1}$ Observed failure count in the range $[s, t_1]$
- X_t Observed failure count in the range $[1, t]$
- X_{t1} Observed failure count in the range $[1, t_1]$

Testing oriented software reliability analysis

Schneidewind model

The parameters are as follows:

— α Failure rate at the beginning of interval S

— β Negative of derivative of failure rate divided by failure rate (i.e., relative failure rate)

Using Schneidewind model， Suppose：

- Number of failure limited， this is means orthogonal time of failure occur is independence and once a time during limited time interval
- Formula of Failure rate: $\lambda(t) = \alpha \exp(-\beta t)$
- Failure is independence during one limited time；
- Change correct rate is proportional to bugs alive；
- Time interval is constant.

— $X_{s,i}$ Observed failure count in the range $[s, i]$

— $X_{s,t}$ Observed failure count in the range $[s, t]$

— $X_{s,t1}$ Observed failure count in the range $[s, t_1]$

— X_t Observed failure count in the range $[1, t]$

— X_{t1} Observed failure count in the range $[1, t_1]$

Testing oriented software reliability analysis

Using Schneidewind model， Suppose：

- Number of failure limited，this is means orthogonal time of failure occur is independence and once a time during limited time interval
- Formula of Failure rate: $\lambda(t)=\alpha\exp(-\beta t)$
- Failure is independence during one limited time；
- Change correct rate is proportional to bugs alive；
- Time interval is constant.

Testing oriented software reliability analysis

In software testing stage, following data was retrieved and recorded:

- The pre-processing of data.
- **The estimating of the model parameters.**
- The analysis of the parameters.
- Reliability and MTBF **prediction.**

- **Bug that linked to REQ**
- **The repairing span of bug**
- **The affected area of bugs**
- **Bugs related number of test cases**
- **The revision of code due to bugs**
- **The time when bugs were reported**
- **The type of bugs**
- **The severity of bugs**

DATA

Bugs found during each test phase through **test LOG**. Data include number of failure TCs、severity of failure、number of bugs、**Time** of failure occur、Version、time of repairing the bug and ……

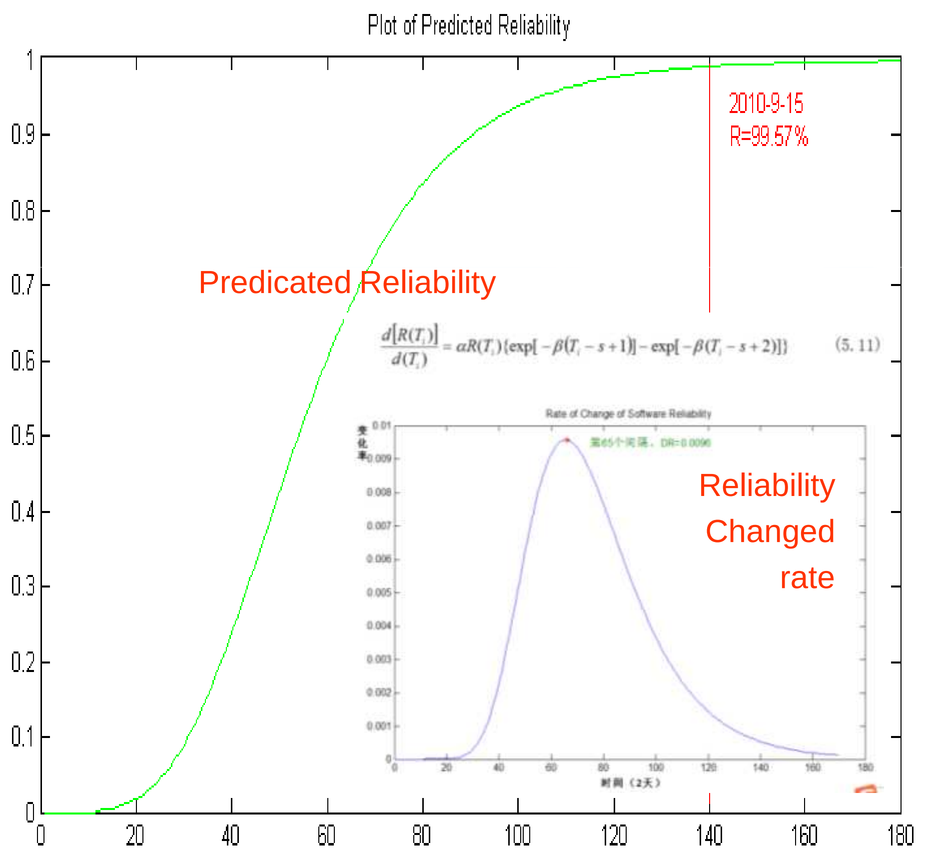
$$\log L = X_t\left[\log X_t - 1 - \log\left(1-e^{-\beta t}\right)\right] + X_{s-1}\left[\log\left(1-e^{-\beta(s-1)}\right)\right] + X_{s,t}\left[\log\left(1-e^{-\beta}\right)\right] - \beta\sum_{k=0}^{t-s}(s+k-1)X_{s-}$$

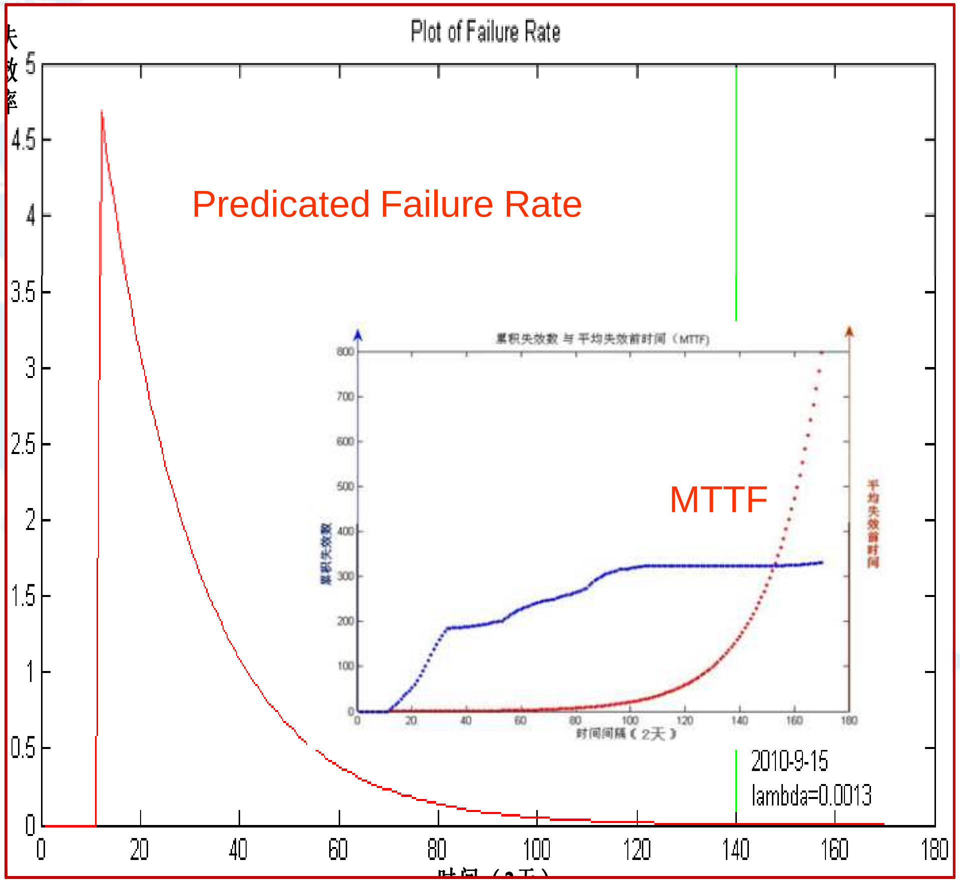
- Using MLE, Assess“α”、“β” parameters of the model iteratively and using WLS/MSE methods to speed up the convergence.
- Determine the time interval “s” parameter.
- More important is to decrease the **effect of study curve** at beginning using mathematical smoothing/average/De-odd etc.

Software Reliability Engineering Practice

$$R(T_i) = \exp(-\frac{\alpha}{\beta}[\exp(-\beta(T_i - s + 1)) - \exp(-\beta(T_i - s + 2))])$$

Reliability Assessment Formula





Trustful analysis result of Reliability based testing

At the review and analysis of test cases, test procedures and test result:

- **Test cases exist for each requirement**
- **The criteria of normal and robustness test are satisfied for each requirement;**
- **The objectives for structure coverage of source code is achieved:**
  - ✓ **Requirement Coverage 100% !**
  - ✓ **Statement coverage 100% !**
  - ✓ **Decision coverage 100% !**
  - ✓ **MC/DC coverage 100% !**

Test is enough from every points of view。
Test procedure DATA is enough for Reliability Assess。
Reliability Trustful

- The distribution of test cases
- The coverage of normal and robustness condition
- Testing intensity sufficiency

4. Discussion

*

- In developing ACoreOS RTOS, ACTRI sum up the answers to problems posed ahead at the kicking off:
  - ✓ How to development reliable software?

    In A/C industry，Strictly follow the processes and objectives quantity defined in **DO-178 level A software**.

  - ✓ How the **predict** or estimate software's reliability?

    Select **Schneidewind reliability model** to predict and estimate reliability in software requirement and testing stage.

Some Questions to be Research :

1、Is it possible to verify the RTOS 100% using several “100%s” coverage ??

2、How to Develop Test Cases according to SRS and meet the 100% coverage ? Is it just the quality of SRS ?

3、Test efficiency and economy ???

4、Is there is more suitable model?

Thanks for Your Attention

Q/A

Aeronautics Computing Technique Research Institute

Aviation Industry Corporation of China