



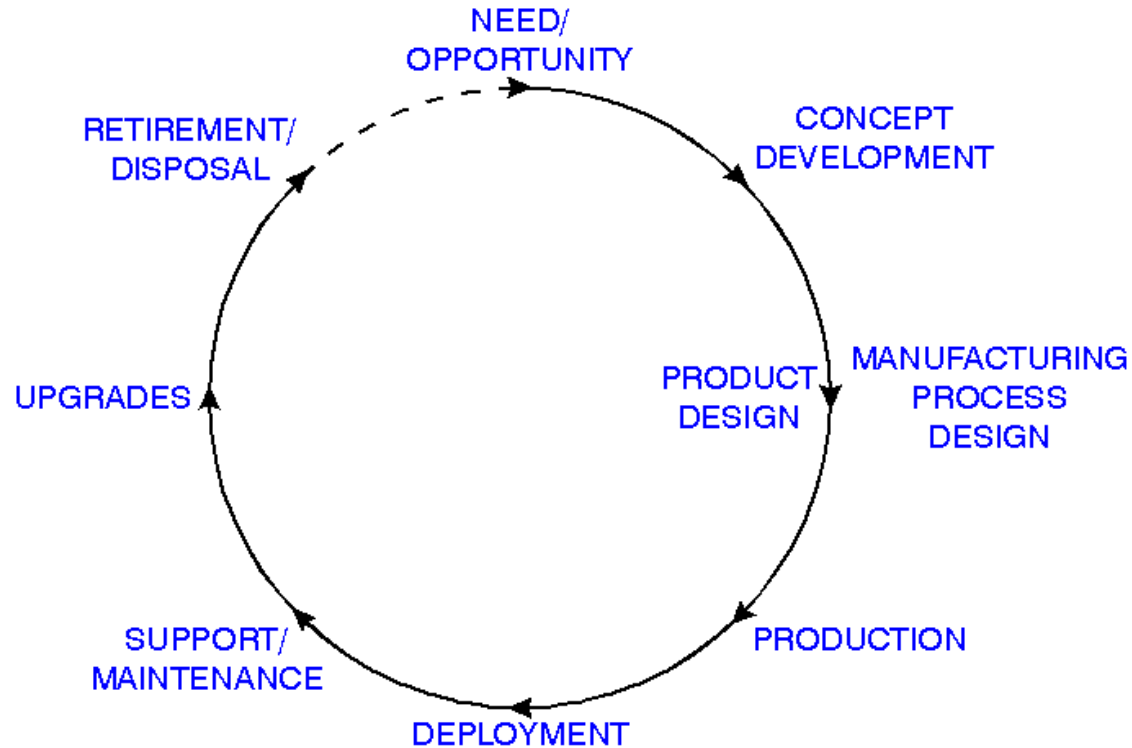
Introduction to Embedded Systems, Lecture 11

# ES lifecycle. Testing, Verification, and Optimization. Debugging ES using simulators

Dmitry Zaitsev

<http://daze.ho.ua>

# Complete Lifecycle of ES



# Reliability of ES

- Testing – run on correct combinations of (input,output) data; difficulty for processes
- Verification – formal proof of correct work – at least correspondence to specifications (requirements)
- Validation is the process of determining that the requirements are correct and complete
- Advantages of using formal systems for ES design: logic, sequential logic, ladder diagrams, Petri nets, etc.

# Real-time systems

- Time constraints (hard and soft)
- Real time clock
- The tasks assigned to real-time systems need to be completed in given time interval
- Performance evaluation techniques
- Optimization of algorithms
- Fast quasi-optimal solutions

# Basic stages of ES Lifecycle

- requirement analysis
- designing
- development
- implementation
- testing
- commercial launch

# Requirement Analysis

- Need
- Target Audience
- Requirements:
  - Operational attributes
  - Functional aspects
  - Product interface components
  - Layout and design preferences
  - Operational and maintenance needs
  - Generic system requirements
- Competitors:
  - A cost-benefit analysis for the project
  - A thorough analysis of the product attributes and functions
  - Product scope and feasibility

# Designing

- Creating the layout and interface for the product
- Software Designing:
  - Identify the features and functionality
  - enhance the interface while keeping it accessible and usable
  - work on a simple and intuitive design with the lowest learning curve.
- Hardware Designing:
  - simple and easy to debug
  - design the hardware for lightweight and smaller sizes
  - fit the devices into the circuits and improve operations
  - the power, the connection, and communication with software

# Development

- Translating the designs into a usable product with proper coding and suitable tools
- Ensure the communication between the hardware and software is well-linked
- Check the development complexities
- Choose the specialists for the projects:
  - Embedded software engineer
  - Hardware engineer
  - PCB layout engineer
  - Mechanical engineer



# Implementation

- Prototype model :
  - Alpha Prototype: It is still in the raw product form, and there could be potential functional issues at this point
  - Beta Prototype: This is the prototype that you get after clearing all the issues and making the product ready for production
- Launch the beta prototype to a closed group of customers to get their feedback and incorporate it into the product

# Commercial Launch

- Approve prototype
- Make product ready for the end-user
- Develop product for commercial use
- Stock the necessary parts and raw materials
- Produce a system
- Sell and deliver system to the end-user

# Testing

- Debugging and testing software
- Debugging and testing ES in simulators
- Debugging and testing hardware-software prototypes of ES
- Testing of ES PCB on dedicated stands
- Complex testing within target system
- Performance/latency benchmarking

# Simulator of Arduino IDE and RP Pico

- <https://wokwi.com/>
- Create and run a project:
  - Arduino IDE sketch – edit code, use libraries
  - Layout of hardware – add and connect parts using graphical editor: controller, sensors, actuators, etc
  - Run project in simulated environment

W pi-pico-community-core.ino - V x +

← → ↺ 🌐 wokwi.com/projects/297322571959894536 ☆ 📁 ⬇️ 👤 ⋮

WOKWI SAVE SHARE ❤️ pi-pico-community-core.ino by urish Docs 👤

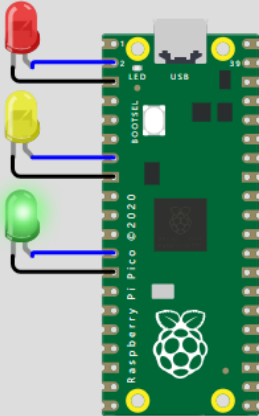
pi-pico-community-core.ino diagram.json Library Manager ▾

```
1 #define RED 1
2 #define YELLOW 5
3 #define GREEN 9
4
5 void setup() {
6   pinMode(RED, OUTPUT);
7   pinMode(YELLOW, OUTPUT);
8   pinMode(GREEN, OUTPUT);
9 }
10
11 void loop() {
12   digitalWrite(GREEN, HIGH);
13   delay(3000);
14
15   digitalWrite(GREEN, LOW);
16   digitalWrite(YELLOW, HIGH);
17   delay(500);
18
19   digitalWrite(YELLOW, LOW);
20   digitalWrite(RED, HIGH);
21   delay(2000);
22
23   digitalWrite(YELLOW, HIGH);
24   delay(500);
25   digitalWrite(YELLOW, LOW);
26   digitalWrite(RED, LOW);
27 }
28
```

Simulation

🔄 ⏏ ⏸

🕒 00:30.785 🔋 99%



🪟 🔍 Type here to search 📁 🌐 📄 🖱️

⤴️ 🖨️ ENG 11:15 AM 4/28/2024 💬

# Realistic Professional Simulator

- Microcontroller virtual machine
- Emulator of electronic circuit
- Emulator of peripheral devices
- Model of controlled system: algebraic equations, ODE, PDE
- Model of faults and breaks
- Benchmarking and analysis

# Black and White Box Tests

- A tester provides an input, and observes the output generated by the system under test.
- Black box testing involves testing a system with no prior knowledge of its internal workings.
- White box testing involves testing an application with detailed inside information of its source code, architecture and configuration.

## Test Strategy

## Tester's View

## Knowledge Sources

## Methods

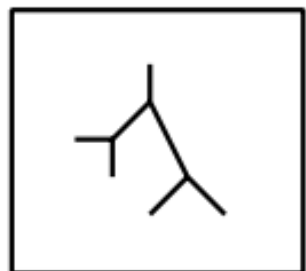
Black box

↓ Inputs



↓ Outputs

White box



Requirements document  
Specifications  
Domain knowledge  
Defect analysis data

High-level design  
Detailed design  
Control flow graphs  
Cyclomatic complexity

Equivalence class partitioning  
Boundary value analysis  
State transition testing  
Cause and effect graphing  
Error guessing

Statement testing  
Branch testing  
Path testing  
Data flow testing  
Mutation testing  
Loop testing



# Black Box Testing Example

Function square\_root (x:real)

when  $x \geq 0.0$

reply (y:real)

where  $y \geq 0.0$  & approximately  $(y*y, x)$

otherwise reply exception imaginary\_square\_root

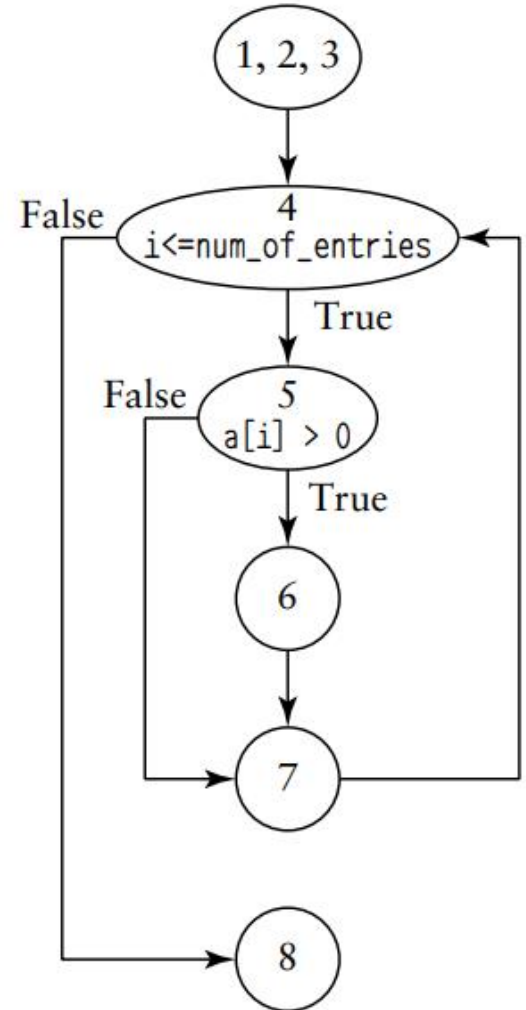
end function

# Black Box Tests

- Equivalence classes:
  - EC1. The input variable  $x$  is real, valid.
  - EC2. The input variable  $x$  is not real, invalid.
  - EC3. The value of  $x$  is greater than 0.0, valid.
  - EC4. The value of  $x$  is less than 0.0, invalid
- Tests:
  - T1:  $2.5+3i \rightarrow$  invalid; EC2
  - T2:  $4.0 \rightarrow 2.0$ ; EC1, EC3
  - T3:  $-5.0 \rightarrow$  exception; EC4

# White Box Testing Example

```
1. pos_sum(a, num_of_entries, sum)
2.     sum = 0
3.     inti = 1
4.     while (i <= num_of_entries)
5.         if a[i] > 0
6.             sum = sum + a[i]
7.         endif
8.         i = i + 1
9.     end while
10. end pos_sum
```



# White Box Test

Decision or branch	Value of variable i	Value of predicate	Test case: Value of a, num_of_entries
			a = 1, -45, 3 num_of_entries = 3
while	1	True	
	4	False	
if	1	True	
	2	False	

# Verification

- Formal proof of correctness with respect to requirements
- Formal methods: logic, automata, discrete-event systems, CSP, Petri nets, LTL, etc
- Verification of software techniques: static and dynamic verification
- Complexity evaluation, proving timed constraints: time and space complexity of algorithms and programs
- Optimization – formulate a criterion and constraints, resolve optimization task: linear and non-linear optimization, discrete optimization

# Static verification of C code

- Automatic analysis for C and C++ projects
- <https://clang-analyzer.llvm.org/>
- Formal proof of program function
- <https://www.key-project.org/>
- Use additional specification written in special language
- <https://github.com/nmaehlmann/verify-c>
- Each function is annotated with logical pre- and postconditions, which specify the contract of the function

# Verification example: loop invariant

- The loop invariant must be true:
  - before the loop starts
  - before each iteration of the loop
  - after the loop terminates

```
int j = 9;
```

```
for(int i=0; i<10; i++) j--;
```

- Loop invariants:
  - $i + j == 9$
  - $i \geq 0 \ \&\& \ i \leq 10$

# Performance evaluation

- Time and space complexity of algorithms and programs
- Dependence on input data size, say  $n$
- Asymptotic evaluation, when  $n \rightarrow \infty$
- “Big-O” notation:  $f(n) = O(g(n)), n \rightarrow \infty$  iff  $|f(n)| \leq C \cdot |g(n)|$  for all  $n \geq n_0$
- Best, average, **worst**



# Minimal element index

```
int mini(int *a, int m){  
    int i, im=0;  
    for(i=1;i<m;i++)  
        if(a[i]<a[im]) im=i;  
    return im;  
}
```

- $M(m) = 2 + 4 \cdot (m - 1) + 1 = 4m - 1$
- $M(m) = 4m - 1 = O(m)$

# Sorting by sequential minimums

```
int sortm(int *a, int n){  
    int i,im,w;  
    for(i=0;i<n-1;i++){  
        im=i+mini(a+i,n-i);  
        w=a[i];  
        a[i]=a[im];  
        a[im]=w;  
    }  
}
```

- $S(n) = 1 + \sum_{i=0}^{n-2} (6 + M(i)) =$   
 $1 + \sum_{i=0}^{n-2} (6 + (4i - 1)) = 1 + \sum_{i=0}^{n-2} (4i + 5)$
- $S = r \cdot (af + al) / 2$ , *sum of arithmetic sequence*
- $S(n) = 1 + (n - 1) \cdot (5 + (4(n - 2) + 5)) / 2 = 1 + (n - 1) \cdot (4n + 2) / 2 = 1 + (4n^2 - 6n + 2) / 2 = 2n^2 - 3n + 1$
- $S(n) = O(n^2)$

# Time complexity comparison

Complexity	Formula	Example	$n = 10$	$n = 20$	$n = 100$
Constant	$c$	10	10	10	10
Logarithmic	$\log n$	$\log_2 n$			
Linear	$n$	$n$	10	20	100
Square	$n^2$	$n^2$	100	400	$10^4$
Polynomial	$n^c$	$n^3$	1000	8000	$10^6$
Exponential	$c^n$	$2^n$	1,024	$\approx 10^6$	$\approx 10^{30}$

Supercomputer Frontier ,  $\sim 10^{18}$  FLOPS, will work  $10^{12}$ s or  $\sim 31000$  years

**infeasible !!!**

# Benchmarks – Measurements on Running System

```
#include <stdio.h>
#include <time.h>
#include <sys/time.h>
```

```
double magma_wtime( void )
{
    struct timeval t;
    gettimeofday( &t, NULL );
    return t.tv_sec + t.tv_usec*1e-6;
}
```

```
main(){
    int *a, n;
    double t1, t2;

    aread(a,&n);
    t1=times();
    sortm(a,n);
    t2=times();
    printf(“%lf s\n”,t2-t1);
}
```

# Optimization of ES

- Optimization task:
  - criteria to minimize (maximize)
  - constraints to satisfy
- Basic ES parameters: response time, power consumption, cost, size, working temperature range, etc
- Example of optimization: knapsack problem – maximize value under the total weight limitation

# Optimization techniques

- Continuous:
  - linear programming
  - gradient descend
- Discrete:
  - exhaustive search
  - greedy strategy (also multistep)
  - branches and bounds
- Dynamic programming (PD) – performance breakthrough

# Dynamic Programming Plot

- Collection of subproblems
- Process them one by one
- Store intermediate results in a table
- From small to large problems
- Use (and reuse) previously tabulated results
- Recover a solution from table

# Optimize Matrix Multiplication

$$M = M_1 \cdot M_2 \cdot \dots \cdot M_n$$

$A_i$  matrix of  $r_{i-1}$  rows and  $r_i$  columns:  $r_{i-1} \times r_i$

$$A = B \cdot C, a_{i,j} = \sum_{k=1}^q b_{i,k} \cdot c_{k,j}$$

To multiply  $p \times q$  by  $q \times r$   
 $p \cdot q \cdot r$  operations are required

$$M = M_1 \cdot M_2 \cdot M_3 \cdot M_4 = \\ M_1 \cdot (M_2 \cdot (M_3 \cdot M_4)) = (M_1 \cdot (M_2 \cdot M_3)) \cdot M_4$$

$r_i$ : 10,20,50,1,100

**125000 versus 2200 operations**  
**~60 times speed up !**



# Matrix Multiplication – DP Algorithm

Let  $m_{i,j}$  - the minimal cost of calculating  $M_i \cdot M_{i+1} \cdot \dots \cdot M_j$ ,  $1 \leq i \leq j \leq n$

$$m_{i,j} = \begin{cases} 0, & i = j \\ \min_{i \leq k < j} (m_{i,k} + m_{k+1,j} + r_{i-1} \cdot r_k \cdot r_j), & i < j \end{cases}$$

$$M_i \cdot M_{i+1} \cdot \dots \cdot M_j = (M_i \cdot M_{i+1} \cdot \dots \cdot M_k) \cdot (M_{k+1} \cdot M_{k+2} \cdot \dots \cdot M_j)$$

$m_{i,k}$  minimal cost of  $M' = M_i \cdot M_{i+1} \cdot \dots \cdot M_k$

$m_{k,j}$  minimal cost of  $M'' = M_{k+1} \cdot M_{k+2} \cdot \dots \cdot M_j$

$r_{i-1} \cdot r_k \cdot r_j$  cost of  $M' \cdot M''$

# Matrix Multiplication – DP Example

$$M = M_1 \cdot M_2 \cdot M_3 \cdot M_4$$

$$r_i: 10, 20, 50, 1, 100$$

$m_{1,1} = 0$	$m_{2,2} = 0$	$m_{3,3} = 0$	$m_{4,4} = 0$
$m_{1,2} = 10000$	$m_{2,3} = 1000$	$m_{3,4} = 5000$	-
$m_{1,3} = 1200$	$m_{2,4} = 3000$	-	-
$m_{1,4} = 2200$	-	-	-

# Summary of ES Design

- Electronic circuit and PCB design
- Software design
- Performance evaluation, benchmarking and optimization
- Testing, verification, and validation
- On simulators, on hardware prototypes, on experimental samples, on plant (in field)