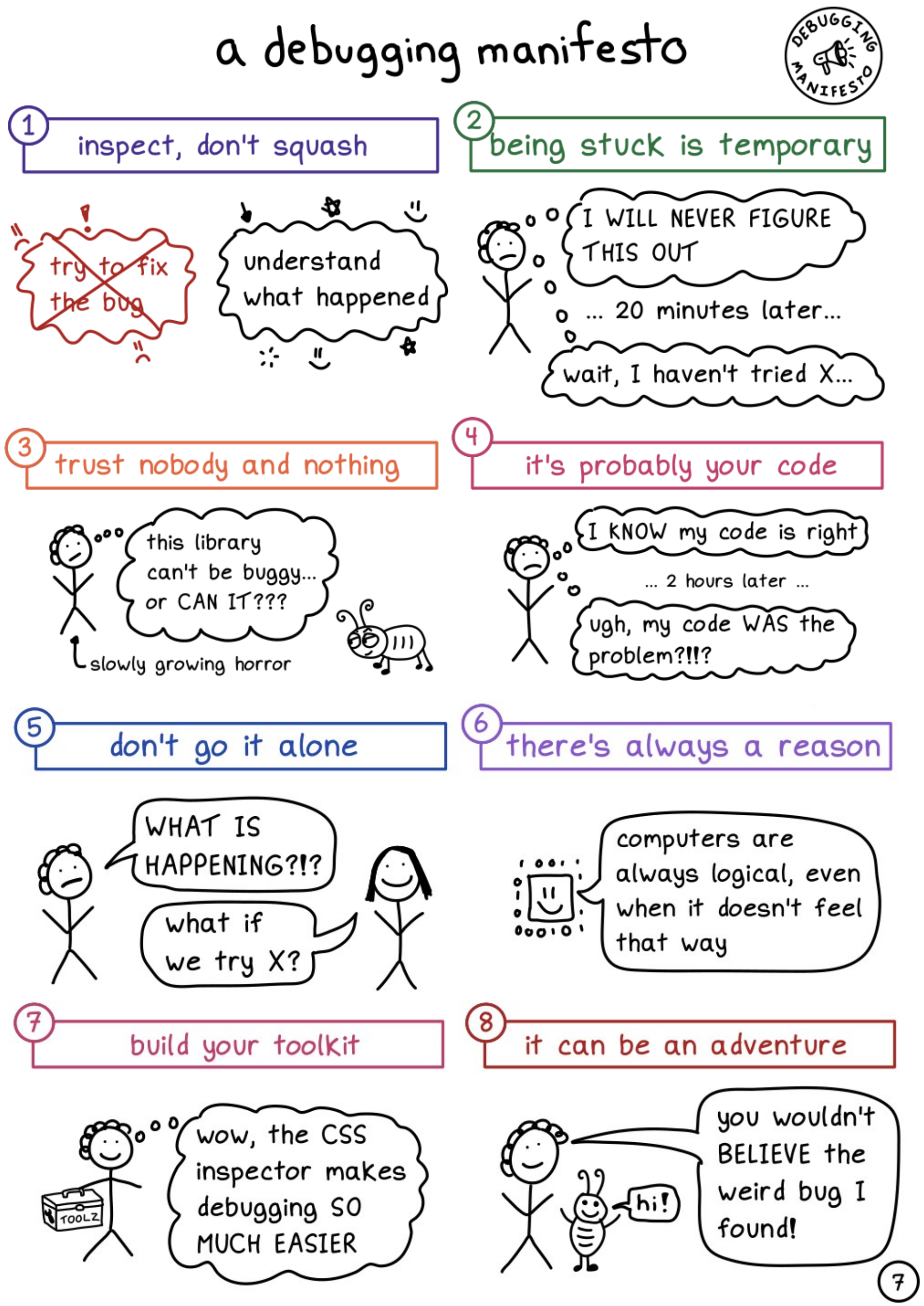
ECCS-3351
Embedded Realtime Applications (ERA)

Debug Real-time Systems

JONATHAN W. VALVANO
Modified by Drs. Kropp, Oun, and Youssfi



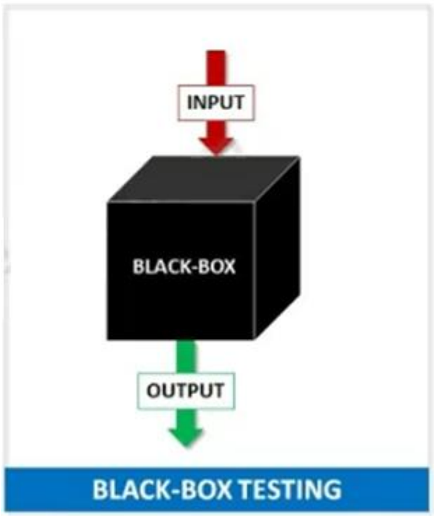
https://wizardzines.com

Debugging Theory

- Every programmer is faced with the need to debug and verify the correctness of his or her software.
- A debugging **instrument** is hardware or software used for the purpose of debugging.
- Hardware-level probes
  - logic analyzer
  - Oscilloscope
- Software-level tools
  - Simulators
  - Monitors
  - Debuggers
  - Manual tools like inspection and print statements.

Testing Method

- **Black-box testing** is simply observing the inputs and outputs without looking inside.
  - Debugs a module for its functionality
- **White-box testing** allows you to control and observe the internal workings of a system.
  - A common mistake made by new engineers is to just perform black box testing.
  - Effective debugging uses both. One must always start with black-box testing by subjecting a hardware or software module to appropriate test-cases.
  - E.g., unit testing



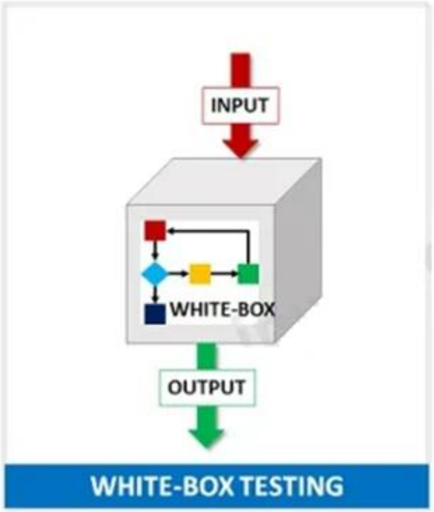
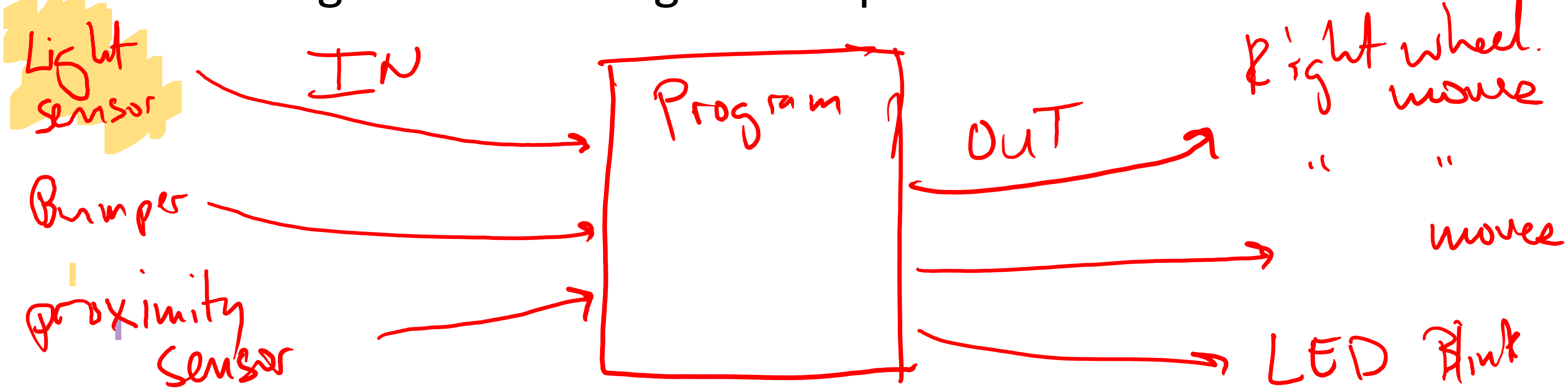


Photo source https://ipwithease.com

Stabilization

- The first step of debugging is to **stabilize** the system.
- In the debugging context, we stabilize the system by creating a **test routine that fixes** (or stabilizes) all the inputs. In this way, we can reproduce the exact inputs over and over again.
- This allows us to precisely measure the effects of the inputs that we want to measure on the outputs
- If some irrelevant input fluctuating uncontrollably, it's confusing what affecting the outputs



Stabilization

- **To stabilize the system** we define **a fixed set of inputs** to test, run the system on **these inputs, and record the outputs**.
- Debugging is a process of finding patterns in the differences between recorded behavior and expected results.

Modular programming

- **To stabilize the system** we define **a fixed set of inputs** to test, run the system on **these inputs, and record the outputs**.
- Debugging is a process of finding patterns in the differences between recorded behavior and expected results.
- The advantage of **modular programming** is that we can perform modular debugging.
- We make a list of modules that might be causing the bug.
- We can then create new test routines to stabilize these modules and debug them one at a time.
- Unfortunately, sometimes all the modules seem to work, but the combination of modules does not.
- In this case we study the interfaces between the modules, looking for intended and unintended (e.g., unfriendly code) interactions.

Intrusiveness

- Most users use manual methods for locating and correcting program errors:
  - Desk-checking
  - Print statements
  - ~~Dumps~~
- A real-time systems cannot be debugged with simple print statements
  - Requires too much time to execute for embedded systems
  - In other words, this is too **intrusive**.

Intrusiveness

- **Intrusiveness** is the measure to which the debugging itself affects the system being measured.
  - **Minimally intrusive** instruments have a negligible effect on the system being debugged.

- Measure of instrustiveness: $t/\Delta t$
  - the fraction of the time consumed by the process of debugging itself
  - t : time to execute the debugging instrument
  - Δt: average time between execution of the debugging tool

- Small enough $t/\Delta t$ = minimally intrusive
  - Must be so small that it's inconsequential to system behavior

$t/\Delta t$ Example

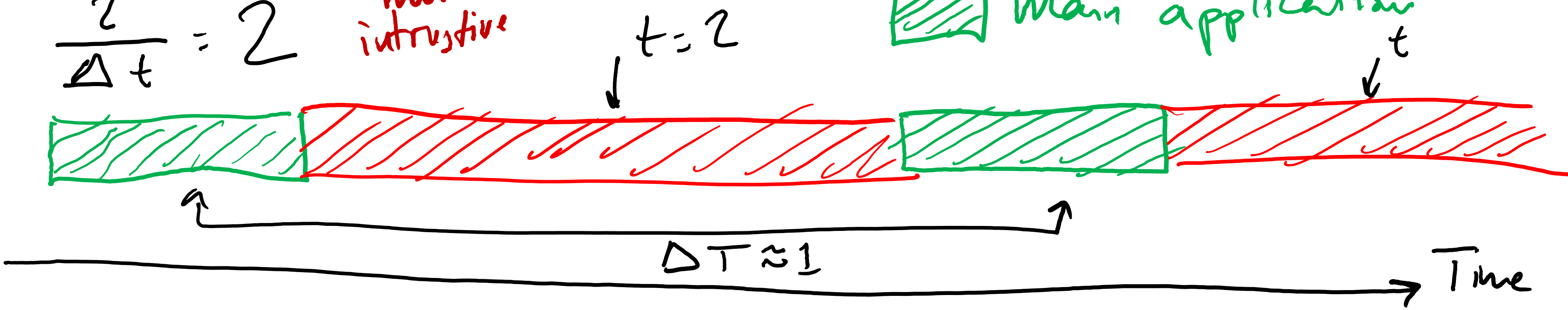
Debugging instrument 1

Instrument running

Main application

$$\frac{2}{\Delta t} = 2$$

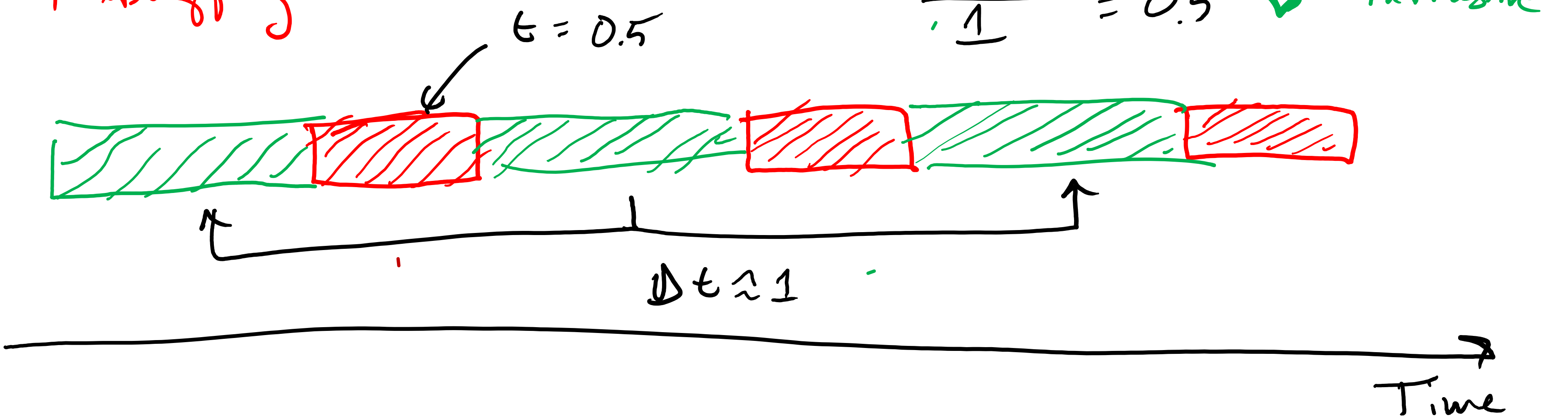
more intrusive



Debugging instrument 2

$$\frac{0.5}{1} = 0.5$$

Less intrusive



Intrusiveness Example 1

- How is this a debugging strategy?
  - `GPIO_PORTF_DATA_R ^= 0x02;`
- This is a heartbeat monitor
- Output pin that turns on and off again
- Is this intrusive or nonintrusive?
  - The heartbeat code requires only 6 bus cycles to execute.
  - If the heartbeat runs every 1ms, and the bus clock is 80 MHz, then is equal to 6/80000. Normally, if this ratio is less than 1/1000 we classify it minimally intrusive

Unintrusive example 2: Dumps

- Memory dumps dump strategic information into an array at run time.
- We can then observe the contents of the array at a later time.
- One of the **advantages** of dumping is that the **debugger** allows you to visualize memory even when the program is running.
- So this technique will be quite useful in systems with a **debugger**.

Dump

- **Intrusiveness:**
  - Short execution
  - Small percentage
    - **Minimally intrusive** if $t/\Delta t$ is small
- **Dump:**
  - Similar usage as printf
  - Save into array (or into flash ROM)
  - Observe later with debugger

```c
#define SIZE 100
uint8_t P1Buf[SIZE];
uint8_t P2Buf[SIZE];
uint32_t I;
void Dump(void){
  if(I < SIZE){
    P1Buf[I] = P1->IN;
    P2Buf[I] = P2->OUT;
    I++;
  }
}
```

Dump example (C)

```
Dump:
00000b08:  48A2    ldr   r0, [pc, #0x288]
00000b0a:  6800    ldr   r0, [r0]
00000b0c:  2864    cmp   r0, #0x64
00000b0e:  D20F    bhs   $C$L1
00000b10:  49A0    ldr   r1, [pc, #0x280]
00000b12:  48C6    ldr   r0, [pc, #0x318]
00000b14:  4AC4    ldr   r2, [pc, #0x310]
00000b16:  6809    ldr   r1, [r1]
00000b18:  7800    ldrb  r0, [r0]
00000b1a:  5450    strb  r0, [r2, r1]
00000b1c:  499D    ldr   r1, [pc, #0x274]
00000b1e:  48C5    ldr   r0, [pc, #0x314]
00000b20:  4AC3    ldr   r2, [pc, #0x30c]
00000b22:  6809    ldr   r1, [r1]
00000b24:  7800    ldrb  r0, [r0]
00000b26:  5450    strb  r0, [r2, r1]
00000b28:  499A    ldr   r1, [pc, #0x268]
00000b2a:  6808    ldr   r0, [r1]
00000b2c:  1C40    adds  r0, r0, #1
00000b2e:  6008    str   r0, [r1]
$C$L1:
00000b30:  4770    bx    lr
```

Dump example (Assembly)

```c
start = SysTick->VAL;
Dump(); // from lecture slide
stop = SysTick->VAL;
dT = 0x00FFFFFF&(start-stop)-11;
```

Measuring intrusiveness of dump

Instrumentation: Dump into Array with Filtering

- One problem with dumps is that they can generate a tremendous amount of information.
- If you suspect a certain situation is causing the error, you can add a **filter** to the instrument.
- A filter is a software/hardware condition that must be true in order to place data into the array.
- In this situation, if we suspect the error occurs when another variable gets large, we could add a filter that saves in the array only when the variable is above a certain value.

Dump Instrument

- **Continuous**
  - Saves the last 32 values
  - Wrap index

```
uint16_t Buf[32];
uint32_t I=0;
void Record(uint16_t x){
  Buf[I] = x;
  I = (I+1)&0x1F;
}
```

I = (I+1) % 32;

- **Filtered**
  - Save only on certain conditions
  - Reduces the volume of data to observe

```
void Record2(uint16_t x){
  if(P1->IN&0x01){
    Buf[I] = x;
    I = (I+1)&0x1F;
  }
}
```

then save

Test case selection

- When a system has **a small number** of possible inputs (e.g., less than a million), **it makes sense to test them all**. When the number of possible inputs is large we need to choose a set of inputs. There are many ways to make this choice. **We can select values:**
  - Near the extremes and in the middle
  - Most typical of how our clients will properly use the system
  - Most typical of how our clients will improperly attempt to use the system
  - ~~You know your system will find difficult~~
  - Using a random number generator

↳ Latin hypercube

Version control

- Using tools like Git can help debug a system
- You make commits whenever your code is working
- When a bug arises, you can look back at your previous working commit to find out what went wrong
  - Like a time machine
- If unsure which commit caused the error, a tool like git bisect can help you search for the errant commit.