

Embedded TDD Cycle –The First 3 Years

Markku Åhman and Timo Punkka
Schneider Electric
first.last at schneider-electric.com



Agenda for today:

- Embedded Characteristics
- Unit Testing
- Dual Targeting
- Acceptance Testing Using Simulation
- Testing in Target HW
- Results and Summary

Embedded characteristics

Cross-compilation

Event-driven

RTOS dependencies

Real time constraints

Low level HW dependencies

Limited tools support

Robustness

Low resource budget

Limited user interface

7

+2

150KLOC C

Legacy Code from 10+yrs

RTOS /w tasking

Modular multi-uC

Safety critical (property and human)

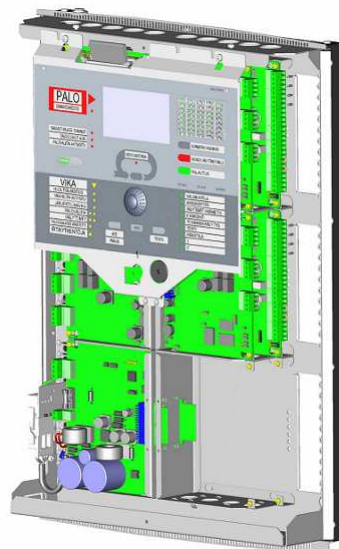
Main Board

Renesas uC

2M Flash

2M RAM

20MHz

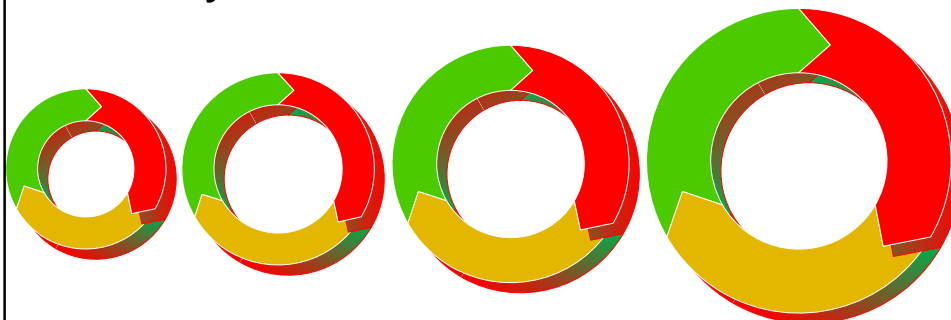




Problem statement:

“We can’t automate the testing of this”

Our Cycles



**Unit Tests in
development
environment**

**Acceptance tests
using simulation**

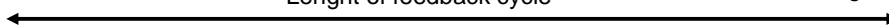
**Automated tests
In target HW**

Explorative Testing

Short

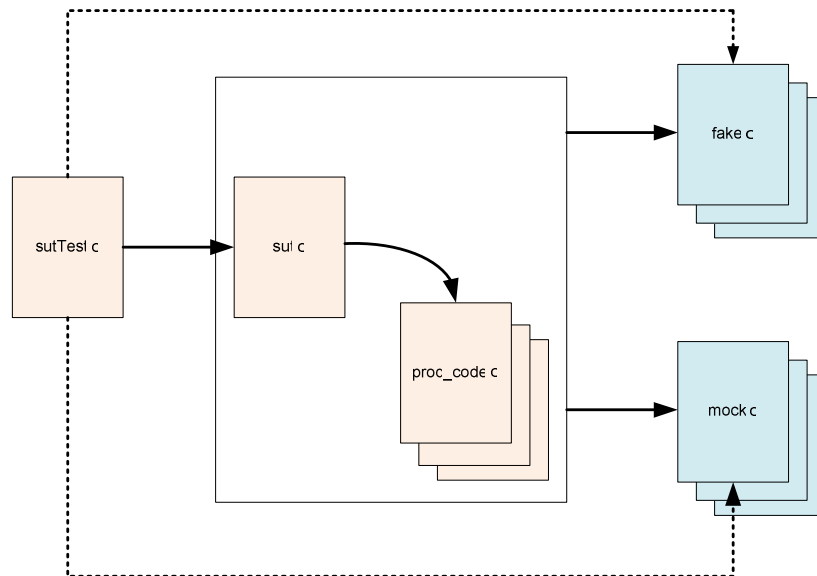
Length of feedback cycle

Long



Adapted from James Grenning, Embedded Test Driven Development Cycle, 2004

Unit Test Build



Test build is described in test file using keywords

```
#include <unity.h>
#include "test/mock/code/loopdb/Mockaddr_alarm_interface.h"
#include "code/utils/lists.h"
#include <something.h>
```

Has worked:

CMock (be careful)

Small group workshops

Higher level tests (through pinch point)

Coaches as visitors

Webinars

Running tests on PC

Dual Targeting

Real
Implementation

```
Main();  
IndicateAlarm();  
DrawAlarmDisplay();  
DrawTextLine();  
DrawCharacter();  
DrawPixel();
```

display memory

Simulation
Implementation

```
Main();  
IndicateAlarm();  
DrawAlarmDisplay();
```

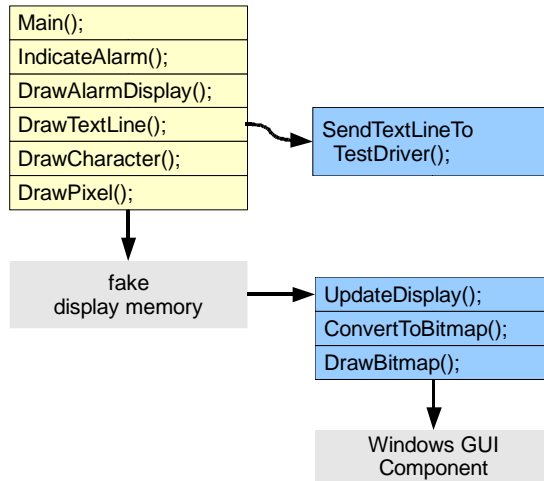
```
DrawTextLine();  
DrawCharacter();  
DrawPixel();
```

display memory

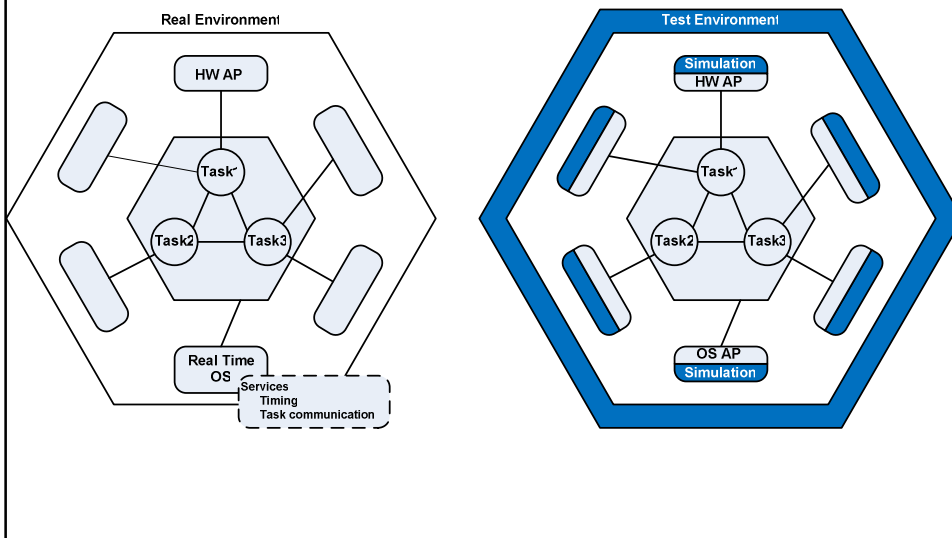
```
DrawTextLine();
```

Windows GUI
Component

Dual Targeting



Dual Targeting



Benefits of Simulation

- Running in development environment
- Fast build-run cycles
- Better debugging environment
- More portable code

Challenges in simulation

- The real-time
- Multi-tasking (OS)
- Distributed systems

Timing multiple executables

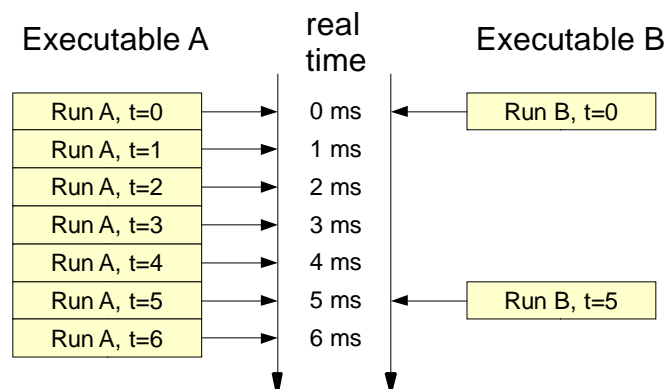
Executable A

```
for (;;) {  
    Run_A();  
    Win32.Sleep(1 ms);  
}
```

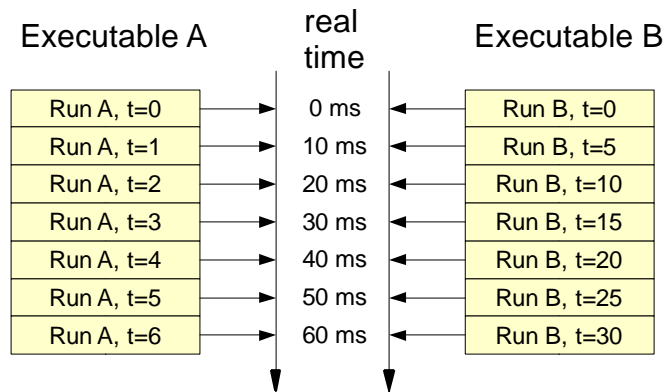
Executable B

```
for (;;) {  
    Run_B();  
    Win32.Sleep(5 ms);  
}
```

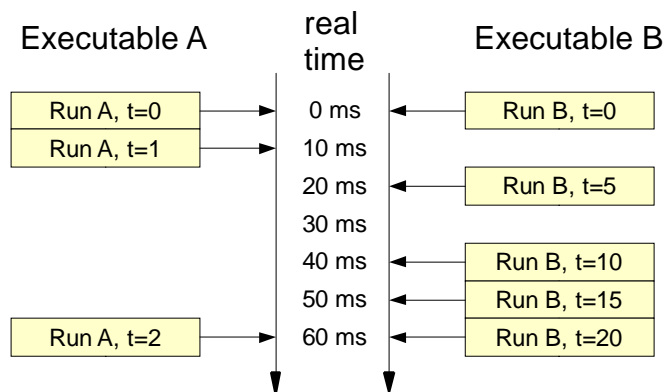
Ideal execution



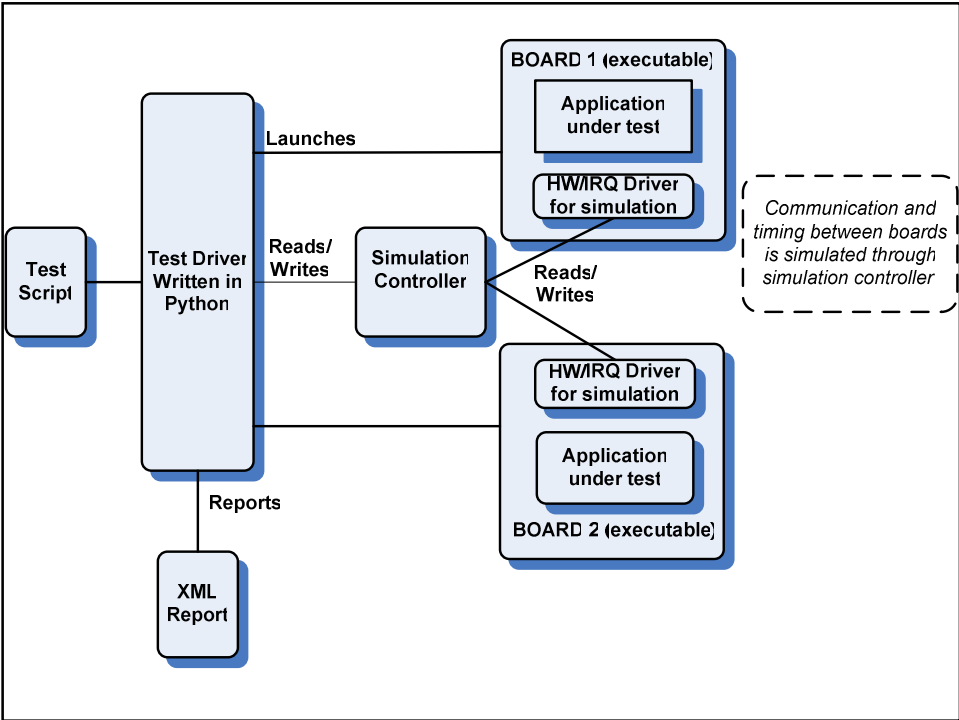
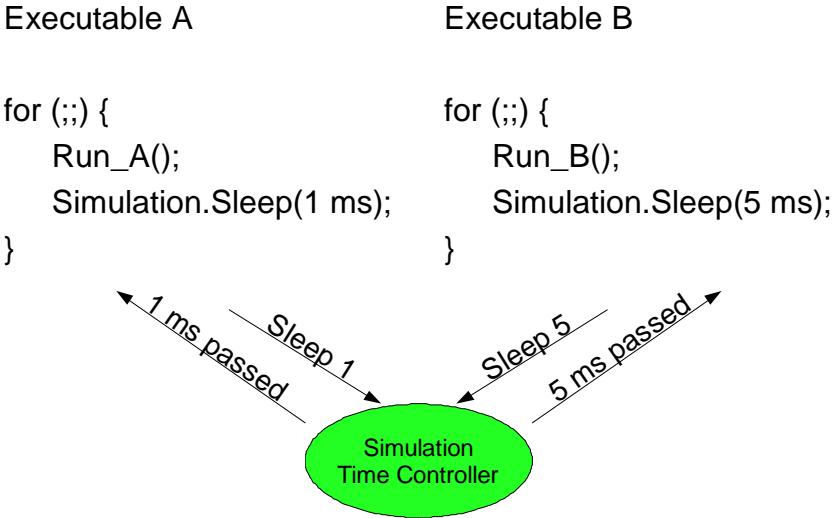
Possible execution

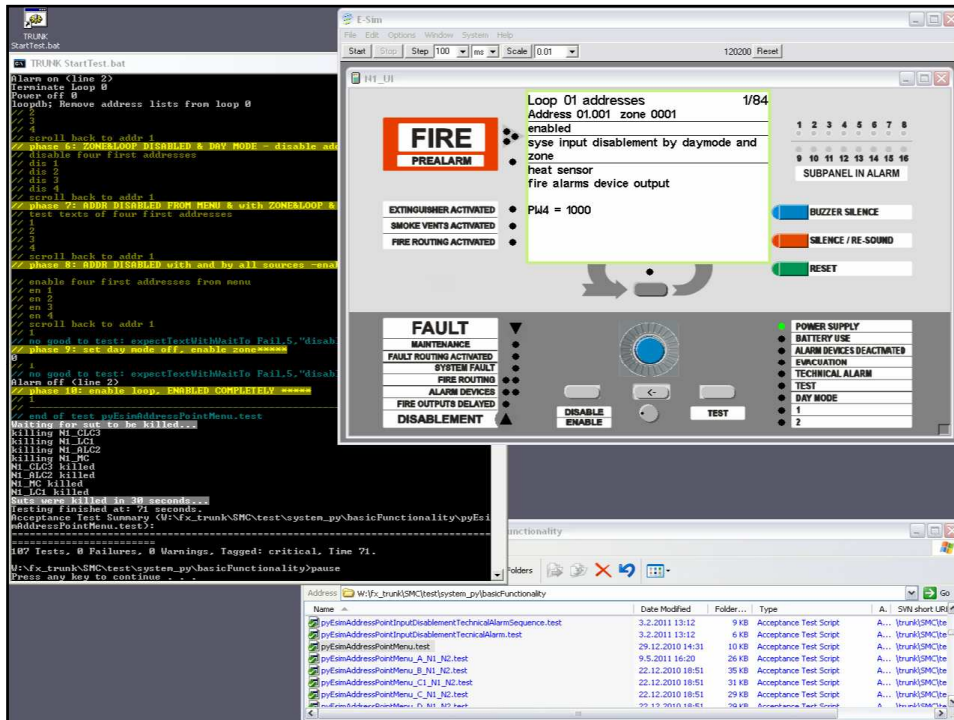


Another possible execution



Simulation time





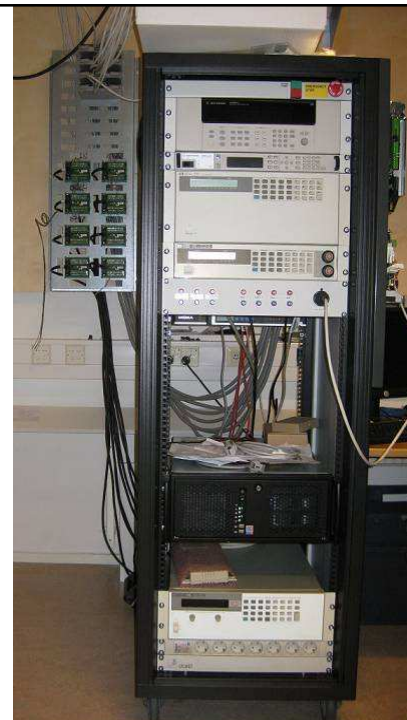
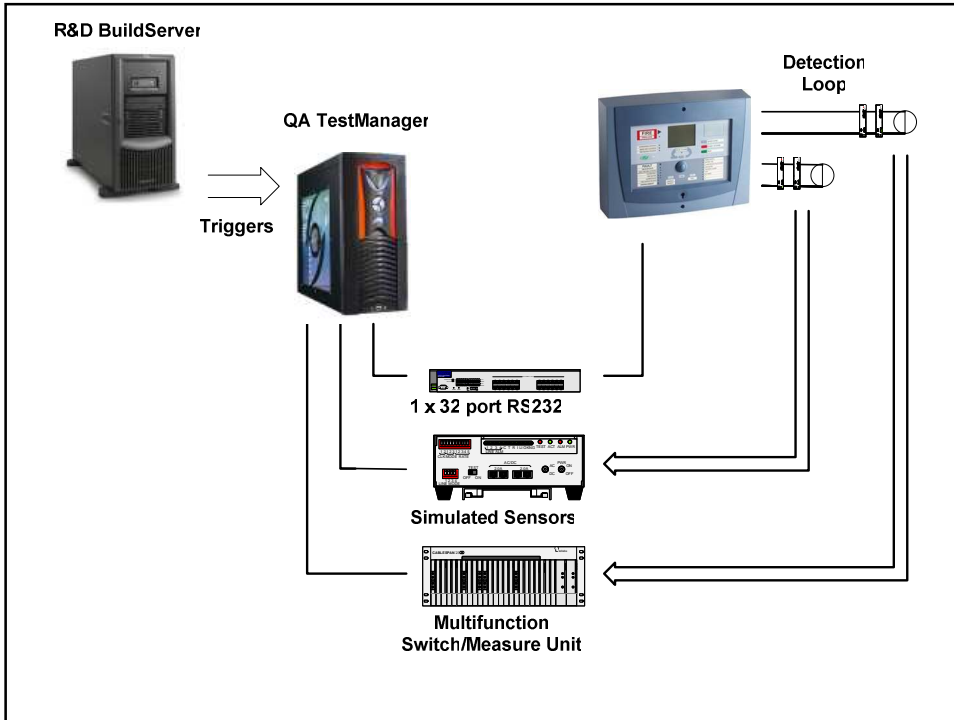
Tested in Development Environment?

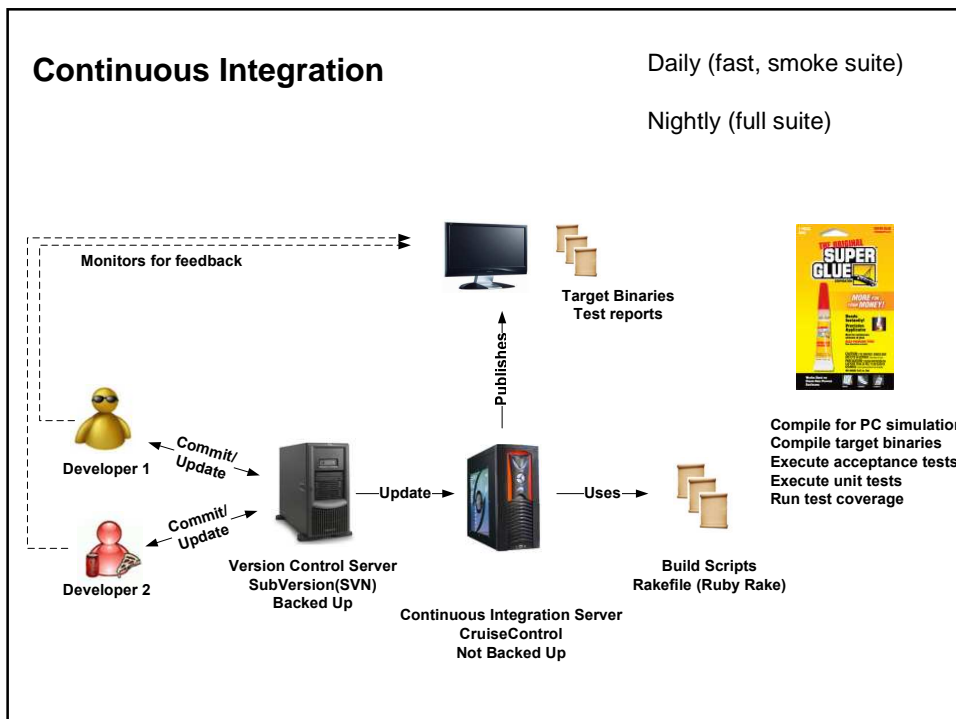
Can:

- High level functionality
- Multi-tasking
- Distributed systems
- Communication of parts
- Communication stress

Cannot:

- Low level functionality
- Real concurrency
- Compiler differences
- CPU stress

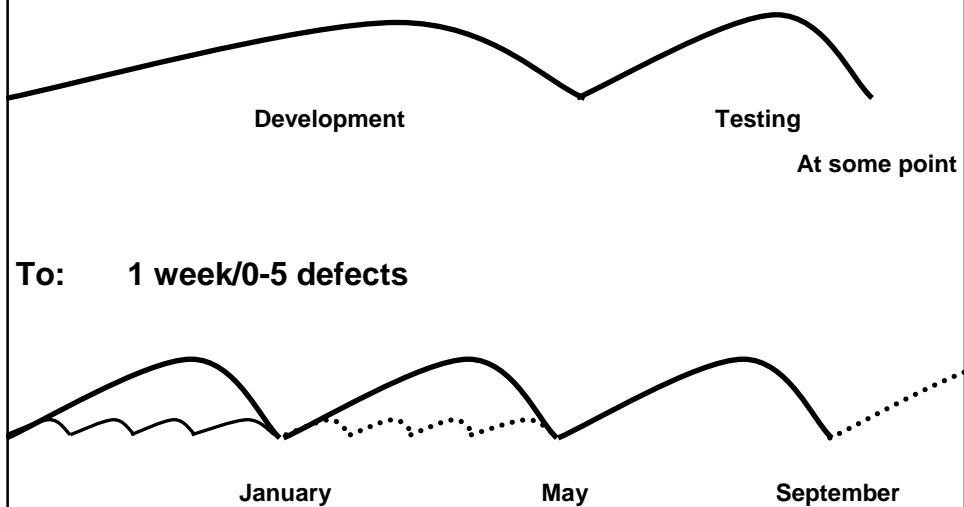




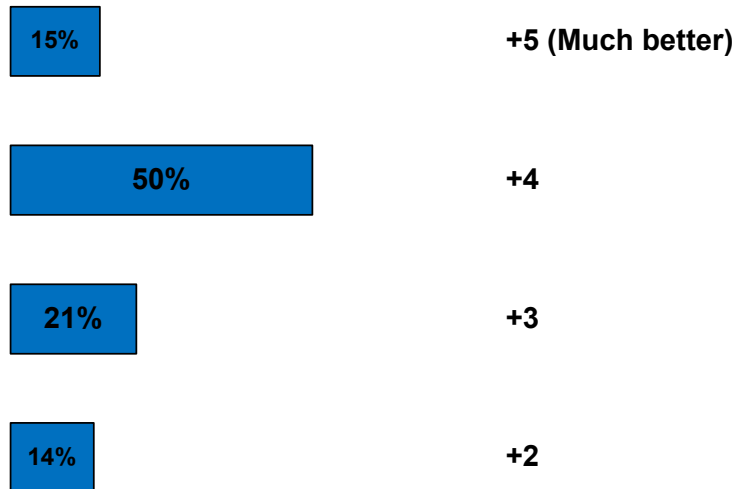
12 releases later...

Final System Test

From: 4 months/150 defects



Survey results (+5 .. 0 .. -5)

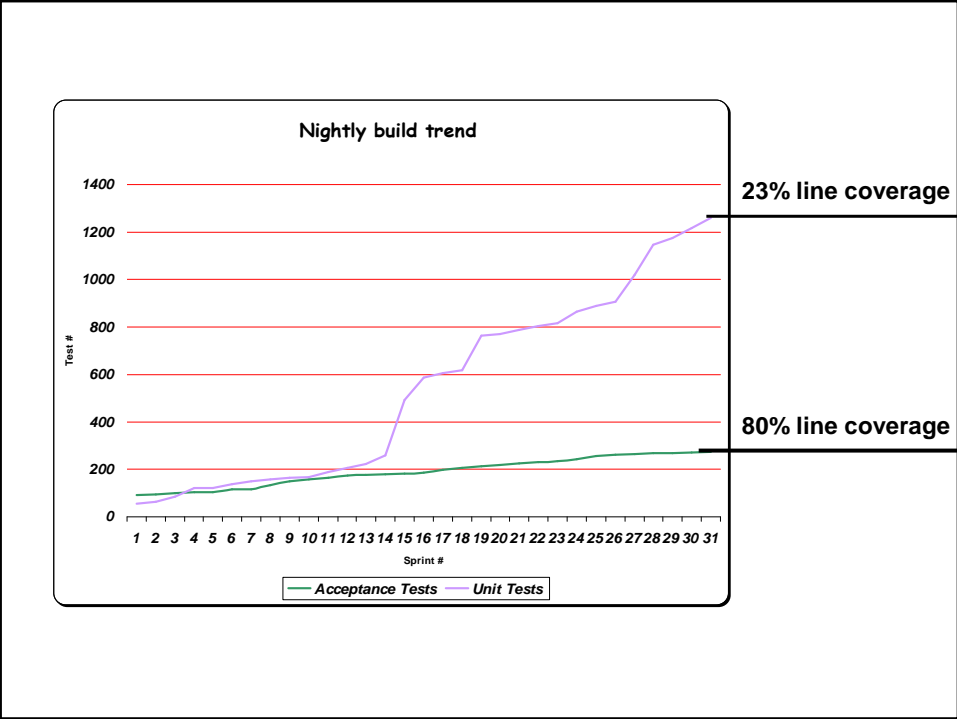


Key Remaining Challenges

Acceptance tests in low level language

Adopt new design style for testability

Flaky tests



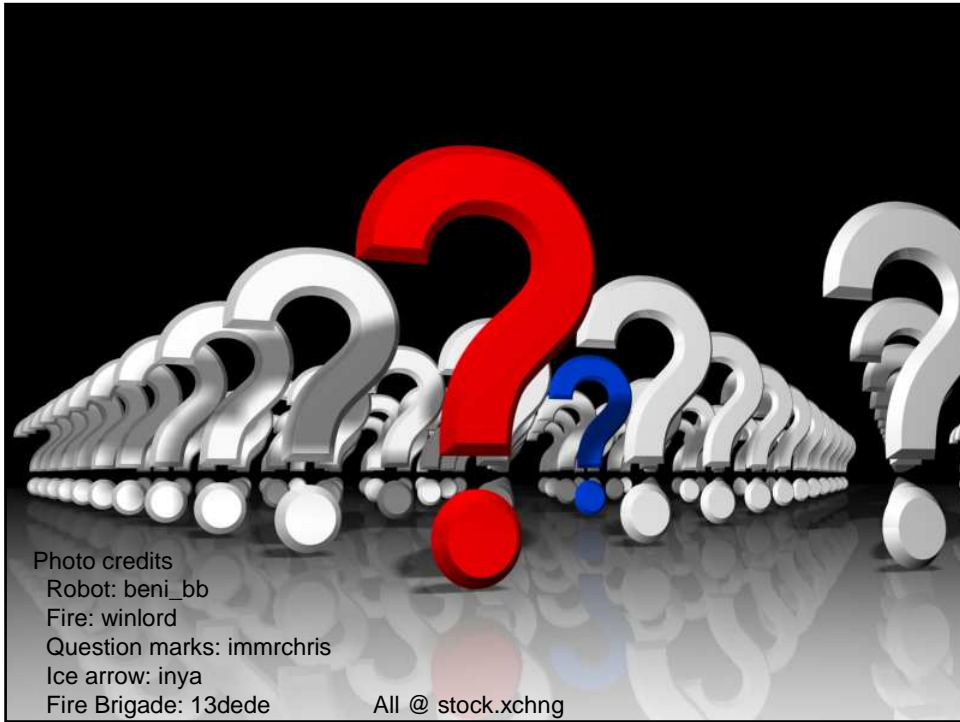


Photo credits
Robot: beni_bb
Fire: winlord
Question marks: immrchriss
Ice arrow: inya
Fire Brigade: 13dede

All @ stock.xchng