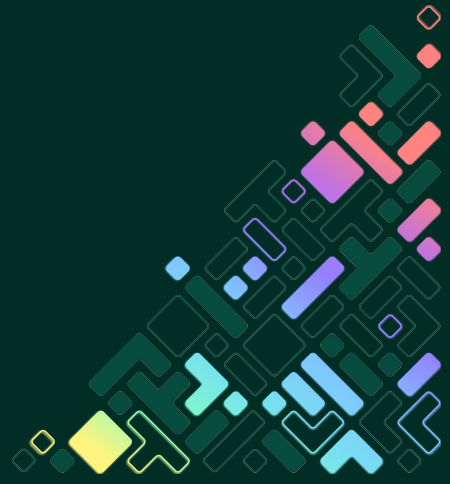
Zephyr® Project
Developer Summit

CROXEL

Application Development with TDD

Getting Started

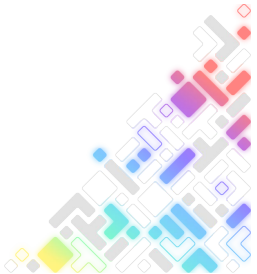
Luis Ubieda
Lead FW Engineer



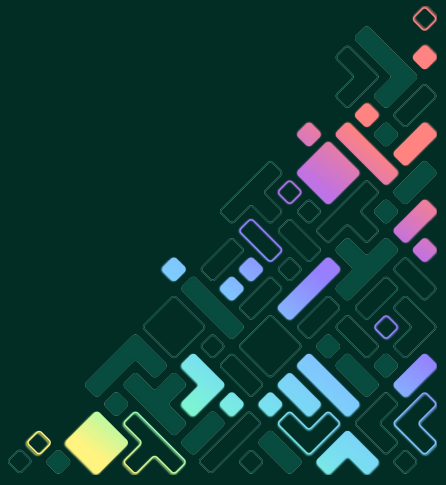
CROXEL

Agenda

- Intro
- Basics of Test-Driven Development (TDD)
- TDD in Zephyr
- Faking dependencies (Device-Tree Nodes and Subsystems)
- Live Demo
- Q&A



Intro



About Me

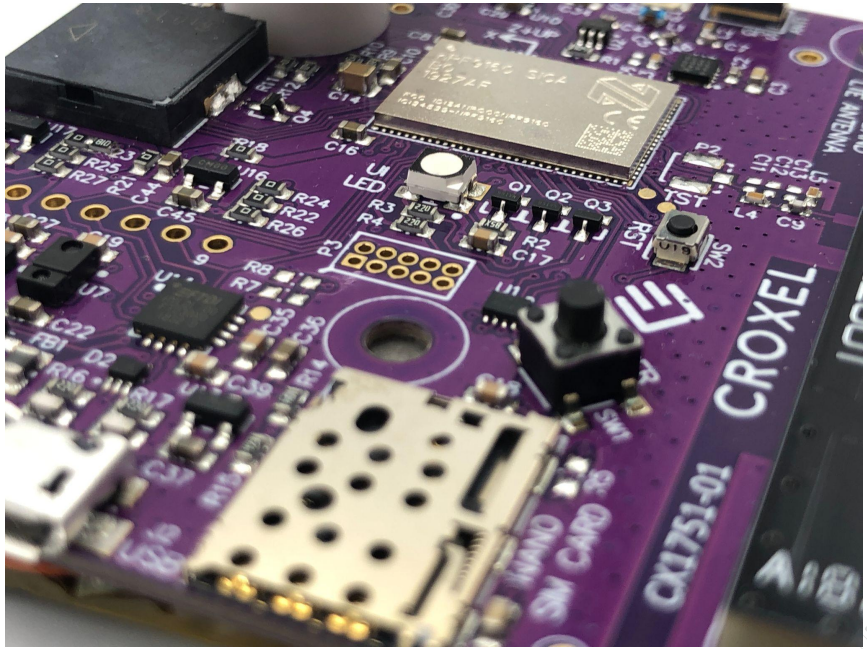
Luis Ubieda

- Electrical Engineer
- Lead Firmware Engineer @ Croxel
- 7 Years developing Firmware
- Based in Melbourne, FL (Space Coast)
- Passion for Technology, Electronics and IoT
- First Zephyr Contribution on 2021
- Free-Time: Fitness and Sports
- Blogger: https://embeddedtales.blog
- First Time Speaker at EOSS

Rocket Launch every other week at the Space Coast.
Image Source: visitspacecoast.com

About Croxel

- Deep Zephyr RTOS expertise for tailored, technically sound solutions.
- End-to-end product development capabilities aligned with your goals.
- Rapid prototyping and low-power optimization for faster time-to-market.
- Seamless integration with your teams across all areas of development.
- Active contributions to the Zephyr Project and continuous community involvement.



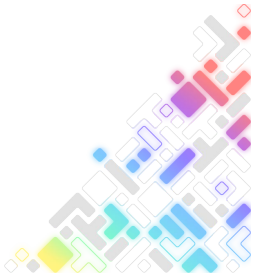
Basics of Test-Driven Development

Basics of Test-Driven Development (TDD)

CROXEL

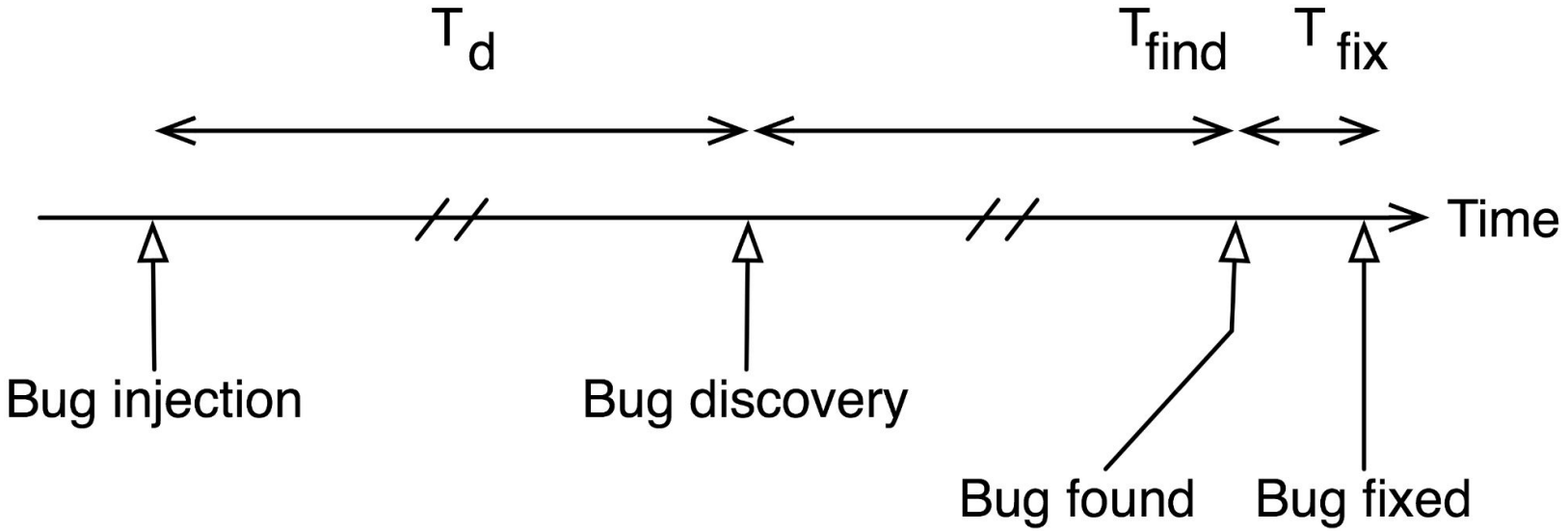
Fundamentals

- Comes from Extreme Programming (XP).
- Opposite to typical Waterfall-type development (develop first, test later).
- Focuses on **incremental** development.
- Embraces the fact that **bugs are inevitable**.
- Relies on capturing and passing **Unit Tests** as means to develop the application.



Basics of Test-Driven Development (TDD)

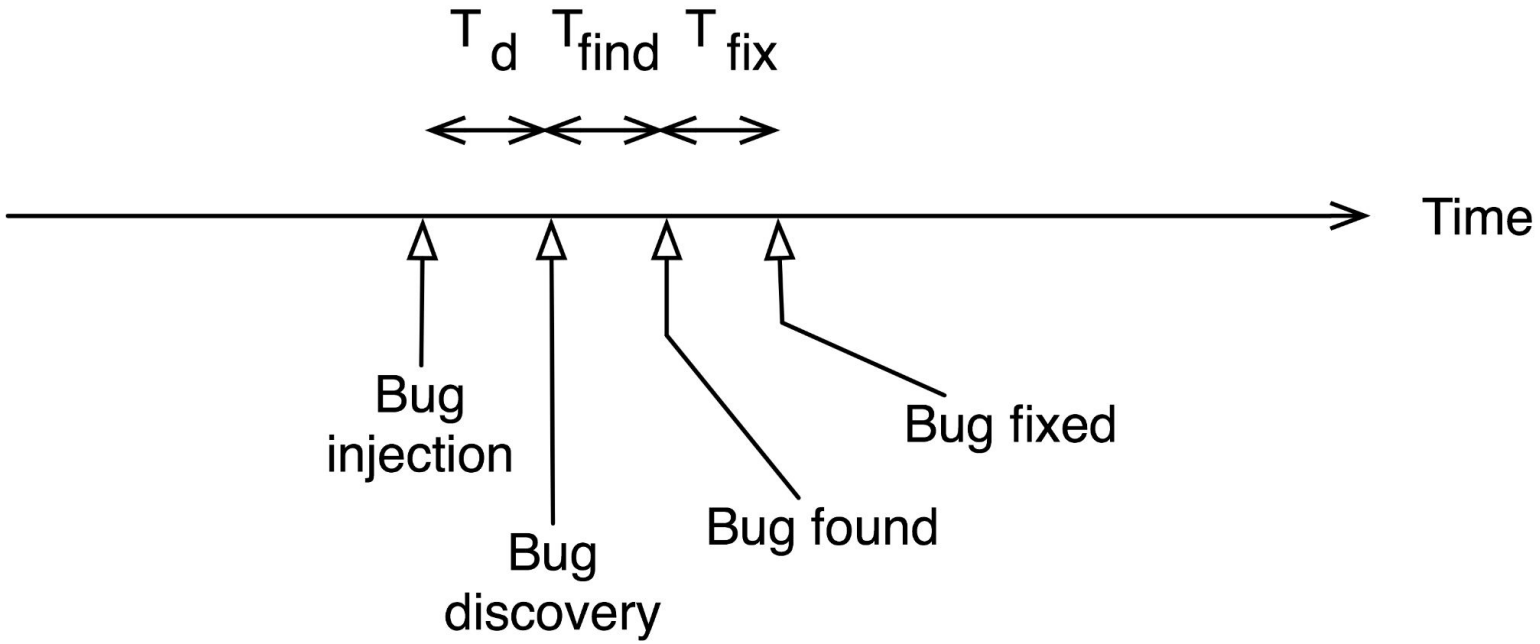
Bugs in a Regular Development Cycle



Source: Test-Driven Development for Embedded C, James Grenning

Basics of Test-Driven Development (TDD)

Bugs in a TDD Development Cycle



Source: Test-Driven Development for Embedded C, James Grenning

Basics of Test-Driven Development (TDD)

Steps to Progress on TDD

- Start with a List of Requirements.
- From requirements to a List of Tests.
- One test at a time.
- Iteration Cycle: **Fail** -> **Pass** -> **Refactor**.
- Capture new test-cases while iterating.

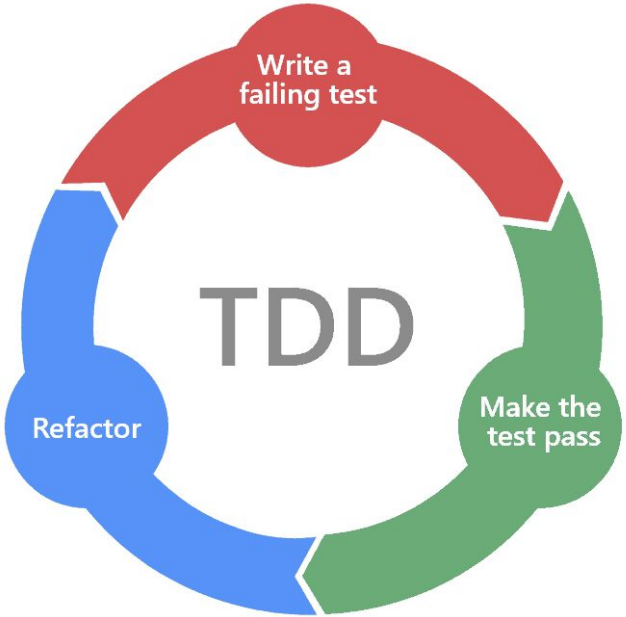


Image Source: https://marsner.com/blog/why-test-driven-development-tdd/

Basics of Test-Driven Development (TDD)

Example Requirements - Motion Detection Engine

1. The Possible Motion states are: Movement, Idle or Unknown.
2. Upon initialization the motion state is Unknown.
3. Idle means no motion has occurred in the last 5 seconds.
4. Movement means detecting at least 2 m/s^2 acceleration on any axis for at least 1 second.
5. A Motion State change triggers an event to users.

Basics of Test-Driven Development (TDD)

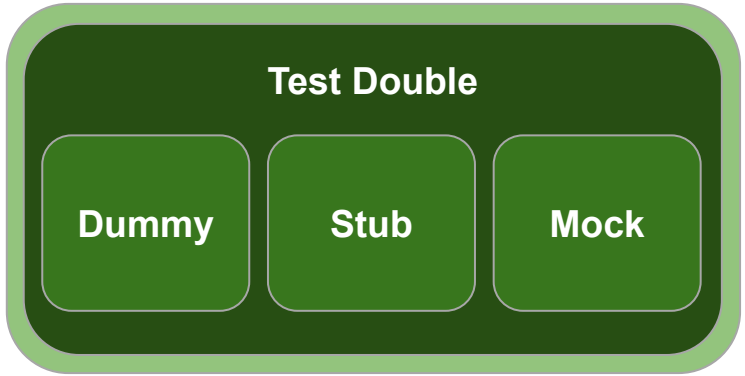
Example List of Tests (Initial) - Motion Detection Engine

1. Initializing the Module starts the Motion Detection Engine.
2. Initial State is Unknown.
3. Callback is triggered on a state change.
4. Transition to Idle state if no motion in 5 seconds.
5. Transition to Movement state if Motion is sustained for 1 second
6. Once entered, stay in Movement state for 5 seconds.
7. Sustained motion while in Movement extends the motion state.

Basics of Test-Driven Development (TDD)

Faking Dependencies

- Unit Under Test (UUT) relies on dependencies.
- Faking a dependency means replacing it by a **Test Double**, abstracted to the UUT**.**
- Test Double sets and manages expectations and returning parameters.
- Types of Test Doubles: Dummy, Stub, Mock.
- It may be done manually or by using Frameworks (e.g: CMock, CppUTest, FFF).

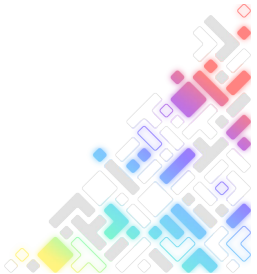


Basics of Test-Driven Development (TDD)

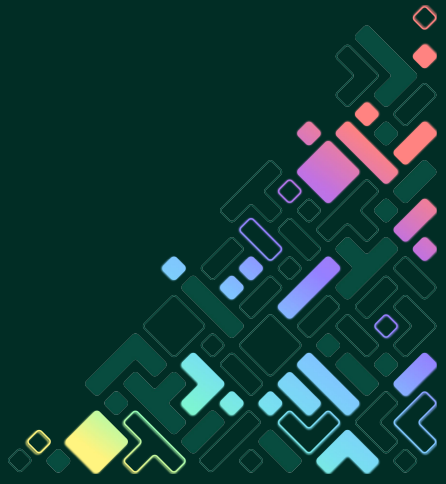
CROXEL

Benefits of TDD

- Software Development without needing Hardware.
- Catch bugs easier and faster.
- Definition of Done is clear.
- Improve Predictability of Development Cycle.
- Code-base feels less “fragile”.
- Overall Improvement of Code Quality.



TDD in Zephyr

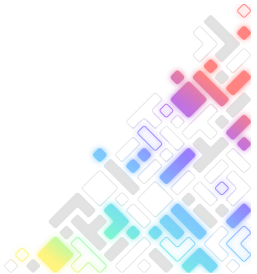


TDD in Zephyr

CROXEL

Testing Frameworks Available

- ZTest, Zephyr Test Framework.
- FFF, Mocking.
- Twister, Test Runner.
- BabbleSim, Radio Simulator (e.g: Bluetooth).
- Pytest, Integration tests.



TDD in Zephyr

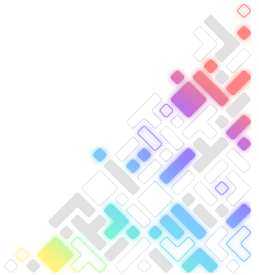
CROXEL

Pros

- Extensive Boards support.
- Inherent Abstractions and Generalized APIs
- Testing Frameworks Available.

Complexities

- Dealing with Device-Tree
- Dealing with Subsystems (Kconfig dependencies).



Faking Dependencies

Faking Dependencies in Zephyr - Subsystems

CROXEL

Alternatives

- Use an existing Zephyr emulator.
- Implement a Test-Double.

Implementing a Test-Double

- Approach: Link-Time Substitution.
- Disable Module using Kconfigs.
- Add minimal Test-Double, implementing the used APIs.

Image Source: https://marketplace.visualstudio.com/items?itemName=trond-snekvik.kconfig-lang

Faking Dependencies in Zephyr - Device-Tree

Alternatives

- Use an existing Zephyr emulator.
- Develop a Device-Tree Node Test-Double.

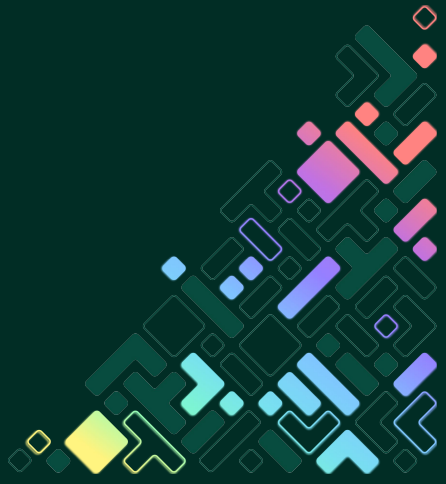
Implementing a DT-Node Test-Double

- Link Device-tree devices using Node-labels.
- Create a DT-Node Test-Double using the same APIs your dependency implements.
- Use the DT-Node Test-Double to emulate the expected behavior (set/check expectations, provide return parameters, etc).

Image Source: https://marketplace.visualstudio.com/items?itemName=trond-snekvik.devicetree

Live Demo

Q&A



Zephyr® Project

Developer Summit