Science & Technology Facilities Council
ISIS

T. Löhnert, F. A. Akeroyd, K. Baker, D. Keymer, A. J. Long,
C. Moreton-Smith, D. Oram (ISIS, STFC)
J. R. Holt, A. McGann, T. A. Willemsen, K. J. Woods (Tessella)

Testing Tools for the IBEX Beamline Control System

Introduction

At ISIS, we are developing the all-purpose next-generation IBEX beamline control system based on EPICS. Developers and users alike need to have full confidence that IBEX works, especially since it is replacing an already fully functional system.

The complexity of a system of this size can be hard to manage, especially with a changing team of developers. To meet this challenge, we require robust tools and processes to ensure correct functionality at all times.

Automated testing is an essential development tool that helps provide us this confidence.

The IOC Test Framework

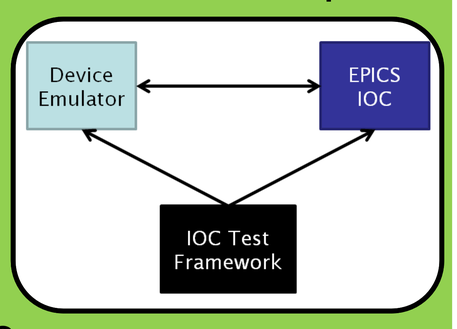
Creating device drivers (*IOCs*) is a large part of what our group does. Hardware is often not available for testing, but errors in driver could lead to lost beam time.

Solution: Framework that tests drivers against device emulators (Python based)

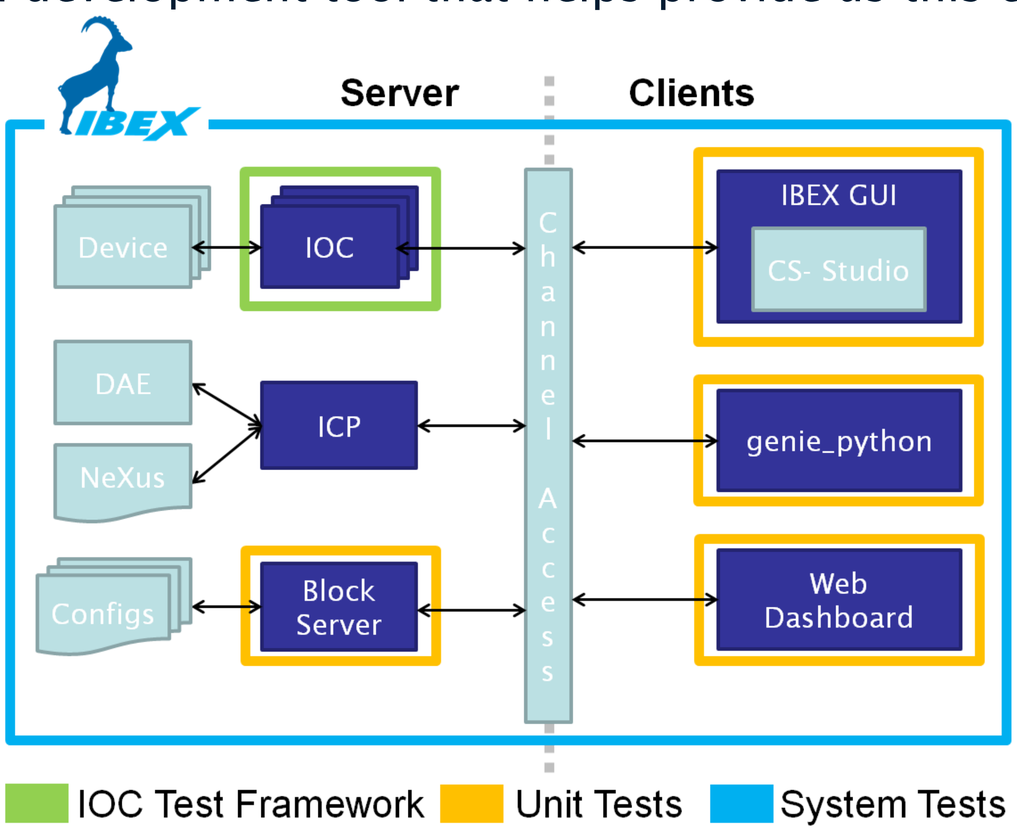
We mostly use emulators written with LeWIS python package (but no dependency)

- Capable of simulating complex stateful devices
- Provides backdoor to device emulator for simulating external events

Example Test: temperature sensor disconnected → stop ramping



1. Start and initialise emulator and IOC
2. Tell IOC to start ramping
3. Assert emulator state is “ramping”
4. Via backdoor, tell emulator to act as if sensor is disconnected
5. Assert both “not ramping”



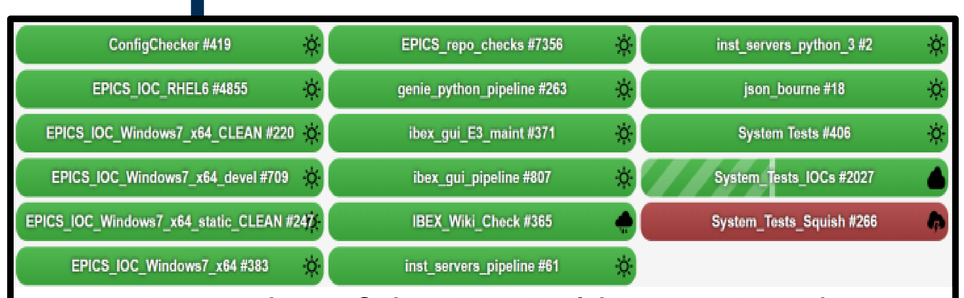
Core IBEX Components

- **IOCs**: EPICS device drivers
- **ICP:** Special IOC, provides Neutron Data
- **Block Server:** Manages beamline configuration
- **IBEX GUI**: CS Studio based main Java client
- **Genie_python:** python scripting command library
- **Web Dashboard:** Read-only web interface to beamline status

This is a selection – IBEX has more components

Continuous Integration

- We use Jenkins platform for CI
- Tests are run every time code is changed & once every night
- Build status are displayed on screen in IBEX office
- Also performs sanity checks: git repositories, wiki spelling, beamline configuration validity



Screenshot of the IBEX Build Status Board

Unit Tests

- Using built-in frameworks (Java JUnit, Python unittest)
- Ensure code is testable by using Test Driven Development, patterns (e.g. MVVM)
- Doubles as documentation: tests demonstrate expected behaviour

System Tests

- Test entire IBEX stack
- 2 test suites interacting through different clients: one via genie_python, one via IBEX GUI
- GUI testing done using Squish: Tool simulating user input
- Automating tests massively sped up release process
- Good for hunting tedious bugs (race conditions, memory leaks)

Future Work

- Increase unit test coverage: tools above were introduced over course of project – new code generally has good testing, but old parts of system can be lacking in places
- Some manual system tests remain which can be automated (balance effort vs. priority)
- Testing user scripts – “dry run” option to expose mistakes before they cause failures and lost time