Session Based Testing

Session-based testing *is a software test method that aims to combine accountability and exploratory testing to provide rapid defect discovery, creative on-the-fly test design, management control and metrics reporting.*

Continuous Delivery

At Native Instruments we have to be able to release every 2 weeks after each sprint, this means we are not able to run regression tests anymore and we also have no more stabilization phase. Session Based testing is a quick way to cover as much areas in the code as possible. It also enables other team members besides the QA staff to get involved in testing and using the application, the more people who test the more defects are found.

Product Component Breakdown

Before starting with Session Based Testing the product needs to be divided into components. A good example can be found in the Maschine Project, small example of Maschine Studio:

Maschine Studio

- **Capacitive Knobs**
  - Screen Overlays / Tag Clouds / Parameter Adjustments
- **Browsing / Loading Filetypes**
  - Both Content and Module Browsers Favourites / Prehear
- **Volume / Level Section Sampling**
- **AutoWrite / Modulation Step Mode**
- **Tap Tempo Button Groups and Group Banks**
- **Pages**
  - Mixer
  - Arrange
  - Plugin
  - Channel
  - Scene
  - Pattern
  - Pad / Keyboard Mode Navigate
  - Duplicate
  - Select
  - Solo
  - Mute / Choke
- **Edit Section w/ Jogwheel Undo / Redo**
- **Pad Colours Transport Area**
- **Playback Recording Modes Grid**
- **Loop Behaviour**
- **Perform Engine**
  - Scale / Chords
  - Note Repeat / Arpeggiator
- **All Shift Functions**

A full overview can be found here:

Components and Functions

When you're testing a component, consider its structure, functions, interfaces & any data that it might process. And keep in mind any appropriate attributes; Is it reliable, intuitive, usable, etc? reset the system between actions.

For each test session choose a component (or a number of components) and a subset of functions to test. Sometimes it makes sense to focus on a single function, other times it might make more sense to take a few functions and test them together, or in parallel.

Test Types

Before you start your test, decide on a test approach. Consider what's appropriate for the component / function that you want to test, or the type of issues you expect to find. Choose one of the following approaches:

Functional Test

- Pick a component and identify the things that this component should do.
- Choose a subset of functions of this component to test.
- Determine how you would know if these functions were working.
- Test each function, one at a time.
- See that each function does what it's supposed to do.

Workflow Test (Single Component)

- Think like a user.
- Conduct a tour through a subset of functions of(usually) a single component.
- Don't reset the system between actions.
- Change the timing and sequencing of these actions, maybe try parallel actions at once.

Workflow Test (Multiple Components)

- Think like a user, but this time string a series of workflows together.
- Conduct a tour through a subset of functions of(usually) a single component.
- Don't reset the system between actions.
- Change the timing and sequencing of these actions, maybe try parallel actions at once.

Negative / Boundary Tests

- Consider the extremes of the component / function that you're testing.
- Does it have an upper or lower limit?
- Think of interesting ways to interact with the software that may cause an error. See how it deals with expected vs unexpected inputs. Within reason; again, think like a user. Consider what kind of "mistakes" they might make.

Stress Tests

- Tests that explore how Maschine behaves under abnormally harsh conditions

Soak Tests

- Tests that explore how the software behaves when run over long periods of time (e.g. check that no audio dropouts occur when running the software over long periods of time)

Claim Checks

- Tests that ensure that the claims that we are making about the release in public are accurate

Structuring Test Sessions

Before you run a test session write a short charter with a clear objective. For an idea of what a charter could look like, check out the following. In short, it should explain what you want to test, how you want to test it, and why.

Example Charters

- Explore loading Groups, Sounds, & FX from Maschine's library browser with Maschine Studio on Windows to find issues with the hardware only workflow.
- Explore page and parameter editing with some Komplete plugins to see if labels can be edited without issue, displayed correctly, and recalled with presets and projects.
- Explore the crossplatform save and recall of thirdparty NKSF presets with Maschine running in standalone to see whether parameter states are recalled correctly in projects across multiple platforms.
- Explore audio routing with groups and sounds to see whether routing configurations are functional and can be saved and recalled correctly with projects.

Guidelines

- Use the Test Session Issue Type in JIRA.
- Set Test Platform(OSX/Windows)
- Set Plugin Mode(vst/standalone/au etc)
- Keep sessions short and focused (~30/60mins).
- Debrief where possible.
- Ask yourself:
    - What kind of issues are you expecting to find
    - How would you know if you found one?
    - How long should it take to find an issue in this area? Consider moving on if you've found nothing unusual after a specified amount of time.
- Link any issues to the test session.

Metrics Test Coverage

When all the test sessions have the correct jira issue type and components, fix version, test platform etc has been set correctly it's easy to get metrics of the test coverage. You could create a filter in Jira looking for the test session issue type and correct fix version and create a pie chart for components, platform, plugin mode etc. Like this you can easily which areas needs more focus.