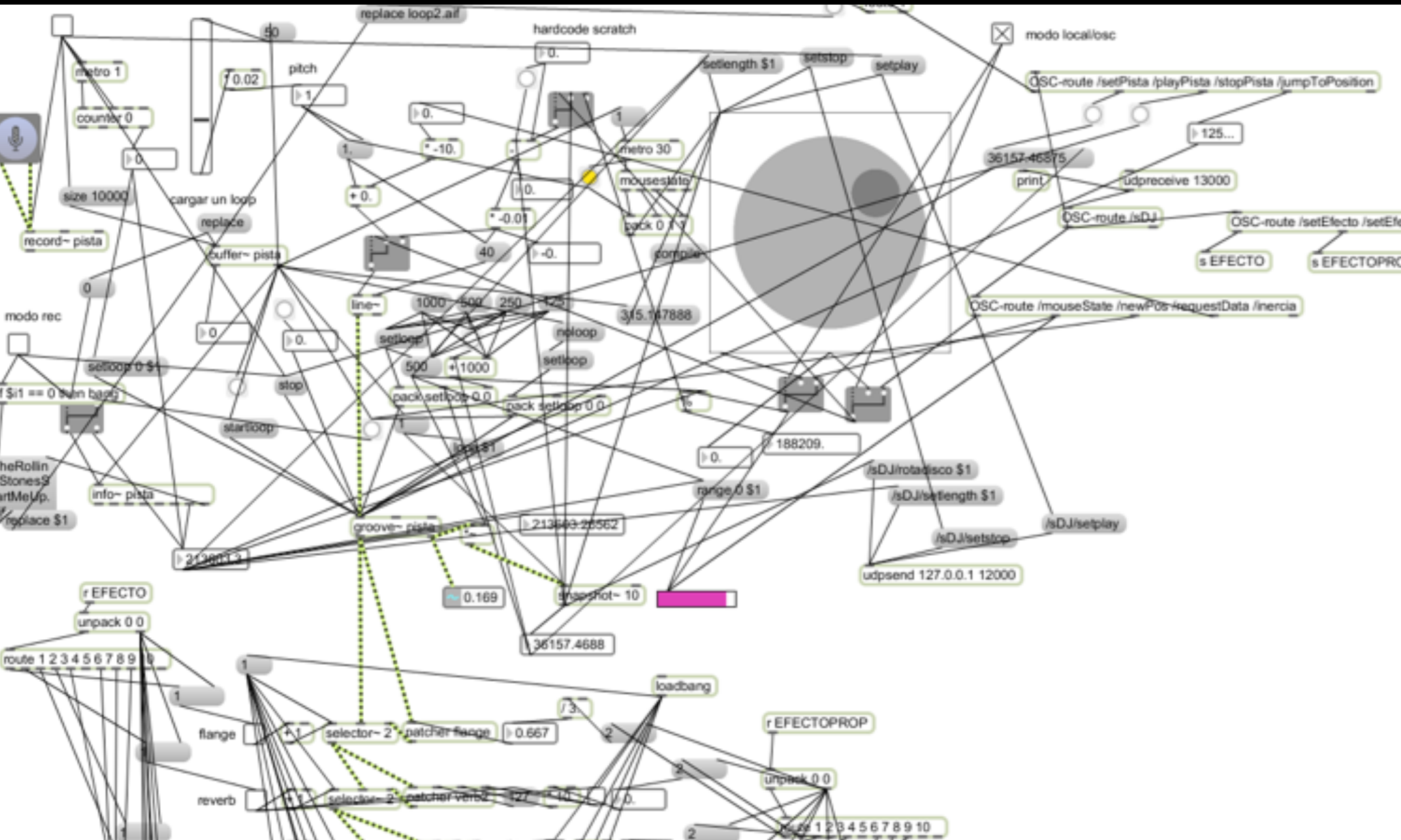
The Jamoma Testing Suite


replace loop2.aif
hardcode scratch
modo local/osc
metro 1
counter 0
pitch
setlength $1
setstop
setplay
OSC-route /setPista /playPista /stopPista /jumpToPosition
size 10000
cargar un loop
replace
record~ pista
buffer~ pista
metro 30
mousestate
pack 0 0 0
compile
print
udpreceive 13000
OSC-route /sDJ
OSC-route /setEfecto /setEfe
s EFECTO
s EFECTOPRO
modo rec
line~
setloop
reloop
OSC-route /mouseState /newPos /requestData /inercia
setloop 0 $1
stop
startloop
info~ pista
replace $1
groove~ pista
range 0 $1
/sDJ/rotadisco $1
/sDJ/setlength $1
/sDJ/setstop
/sDJ/setplay
udpsend 127.0.0.1 12000
snapshot~ 10
0.169
r EFECTO
unpack 0 0
route 1 2 3 4 5 6 7 8 9
loadbang
flange
selector~ 2
patcher flange
0.667
r EFECTOPROP
unpack 0 0
reverb
route 1 2 3 4 5 6 7 8 9 10

Why testing?

We don't need testing, when ....

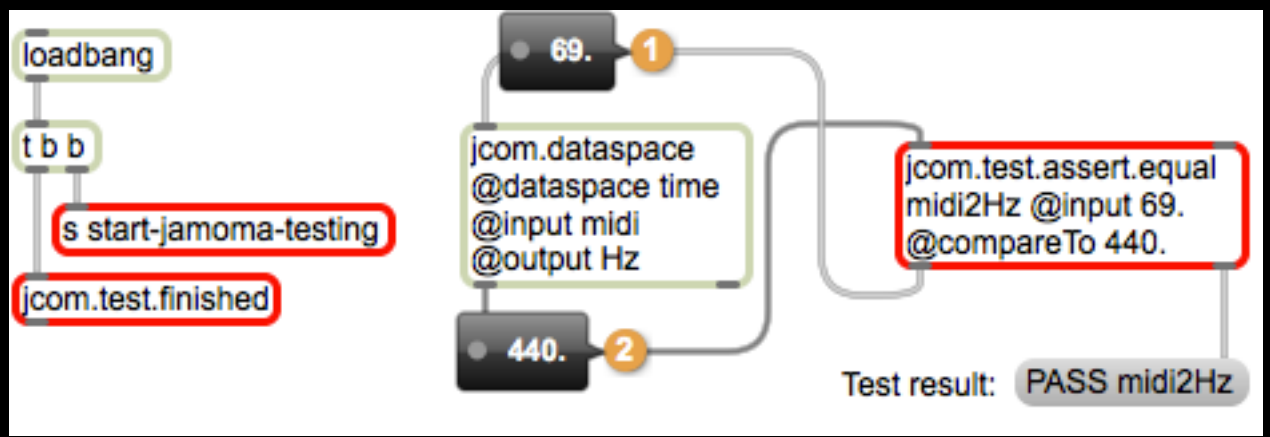
1. ... bugs would not exist
2. ... we don't care about bugs
3. ... we like to be inefficient

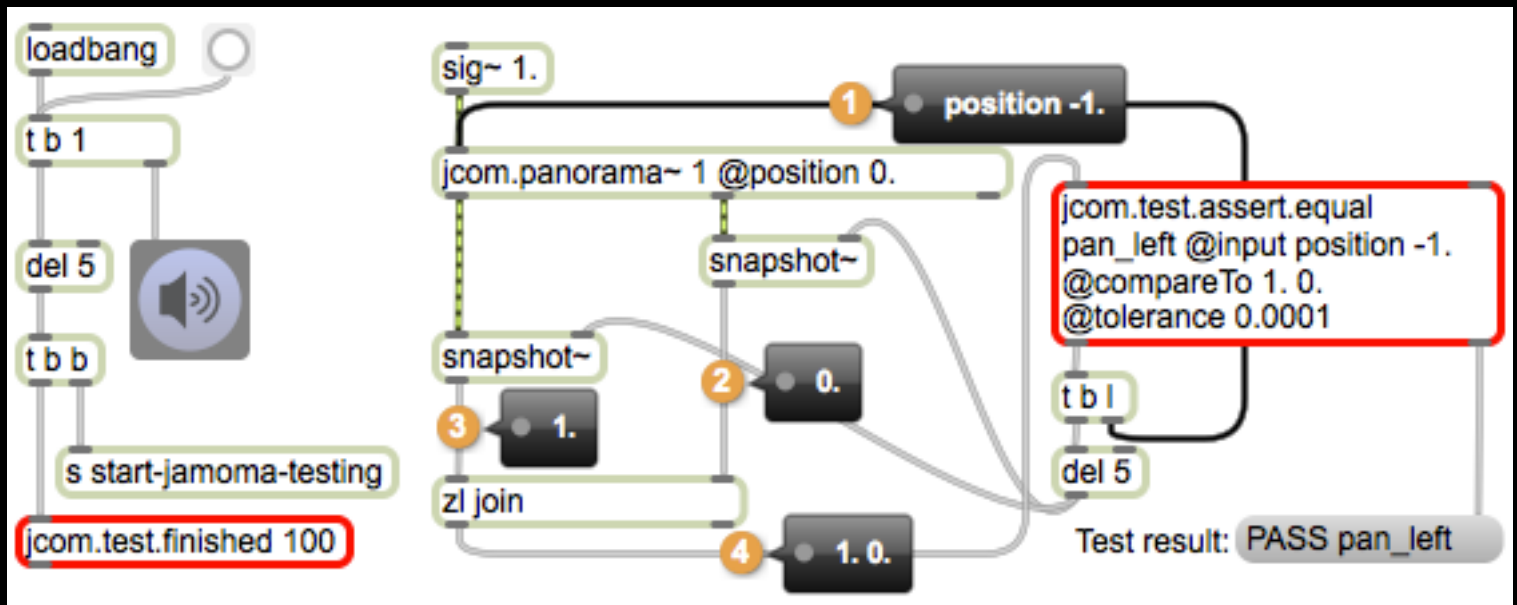
Why automated testing?

- Nobody enjoys testing
- Manual test are prone to errors *(to err is human ...)*
- We don't have the man power for manual testing
- If your patches run on different platforms (Windows, OSX, ??) --> amount of manual testing will multiply
- We don’t want that “cured” bugs reappear
- Some functionalities are even hard for manual testing

Testing tells us...

- What the program is supposed to do
  - This is defined in the specification (ideally)
- What the program is doing
- If the program is doing what it's supposed to


loadbang
t b b
s start-jamoma-testing
jcom.test.finished
69.
1
jcom.dataspace
@dataspace time
@input midi
@output Hz
440.
2
jcom.test.assert.equal
midi2Hz @input 69.
@compareTo 440.
Test result:
PASS midi2Hz


loadbang
t b 1
del 5
t b b
s start-jamoma-testing
jcom.test.finished 100
sig~ 1.
1
position -1.
jcom.panorama~ 1 @position 0.
snapshot~
snapshot~
2
0.
3
1.
zl join
4
1. 0.
jcom.test.assert.equal
pan_left @input position -1.
@compareTo 1. 0.
@tolerance 0.0001
t b l
del 5
Test result:
PASS pan_left

An Automated Testing Suite for Computer Music Environments

Nils Peters
ICSI, CNMAT UC Berkeley
nils@icsi.berkeley.edu

Trond Lossius
BEK
trond.lossius@bek.no

Timothy Place
Electrotap
tim@electrotap.com

ABSTRACT

Software development benefits from systematic testing with respect to implementation, optimization, and maintenance. Automated testing makes it easy to execute a large number of tests efficiently on a regular basis, leading to faster development and more reliable software.

Systematic testing is not widely adopted within the computer music community, where software patches tend to be continuously modified and optimized during a project. Consequently, bugs are often discovered during rehearsal or performance, resulting in literal "show stoppers". This paper presents a testing environment for computer music systems, first developed for the Jamoma framework and Max. The testing environment works with Max 5 and 6, is indepenf from any 3rd-party objects, and can be used with non-Jamoma patches as well.

1. INTRODUCTION

1.1 Testing in sound and music computing

Stability and reliability is a general and important concern in all development and use of software applications. To artists and musicians working with real-time media processing environments such as Max, SuperCollider or Csound, programming is an integral part of their artistic practice. Their patches can be considered software programs, and they also become critical and integrated parts of the resulting artistic works, be that in the form of virtual audio-visual instruments for live performances, or patches used to run installations. In these contexts software reliability is not just a question of being able to work efficiently up front while preparing the artistic work, avoiding the frustrating experience of loosing time and work in progress due to sudden and unexpected bugs and crashes. The very presentation of the works in concerts, performances and exhibitions depends on the software, and quite literally software defects can be show stoppers.

In 2002 the National Institute of Standards and Technology (NIST) reported that software defects cost $59.5 Billion annually in the US, while a third of it could be eliminated by an improved testing infrastructure [1]. One believes that the earlier a bug is found, the cheaper the fix becomes. A systematic approach to testing is part of contemporary programming practice, making extensive use of solutions for running automated tests on a regular basis.

In the sound and music computing community adoption of systematic approaches to testing remain less widespread. Bugs often surface when making changes to the program, or the target environment or operating system, and for this reason many artists tend to be hesitant about changing their performance computer system or software version because of the fear of unforeseen problems. They might also be reluctant to doing last-minute changes and improvements to their patches. The programs or patches developed are often custom developed for a particular project, and with increasing complexity patches become increasingly vulnerable.

1.2 Importance of testing to Jamoma development

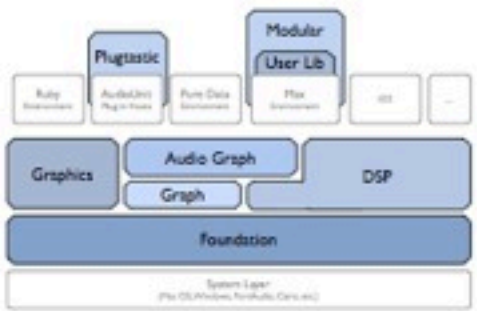


Figure 1. The Jamoma layered architecture

Jamoma is a real-time interactive media processing platform structured as a layered architecture of several frameworks (see Figure 1), providing a comprehensive infrastructure for creating computer music systems [A]. Jamoma Foundation provides low-level C++ support, base classes, and communication systems. Jamoma DSP specializes the Foundation classes to provide a framework for creating a library of unit generators [2]. Jamoma Graph networks Jamoma Foundation based objects into graph structures. Jamoma Audio Graph [3] is a C++ framework that extends and specializes the Jamoma Graph layer. It provides the ability to create and network Jamoma DSP objects into dynamic graph structures for synchronous audio processing. Jamoma Graphics provides screen graphics. Jamoma Modular provides a structured approach to development and control of modules in the graphical media environment Max [4]. Plugtastic, the latest addition to the Jamoma ar-