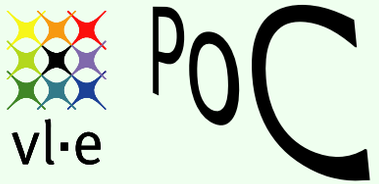
PoC Software Engineering



Dennis van Dok

NIKHEF

VL-e workshop, 7 April 2006, Science Park Amsterdam

Outline

- State of the PoC
- Dependencies and deployment
- Moving ahead to R2

State of the PoC

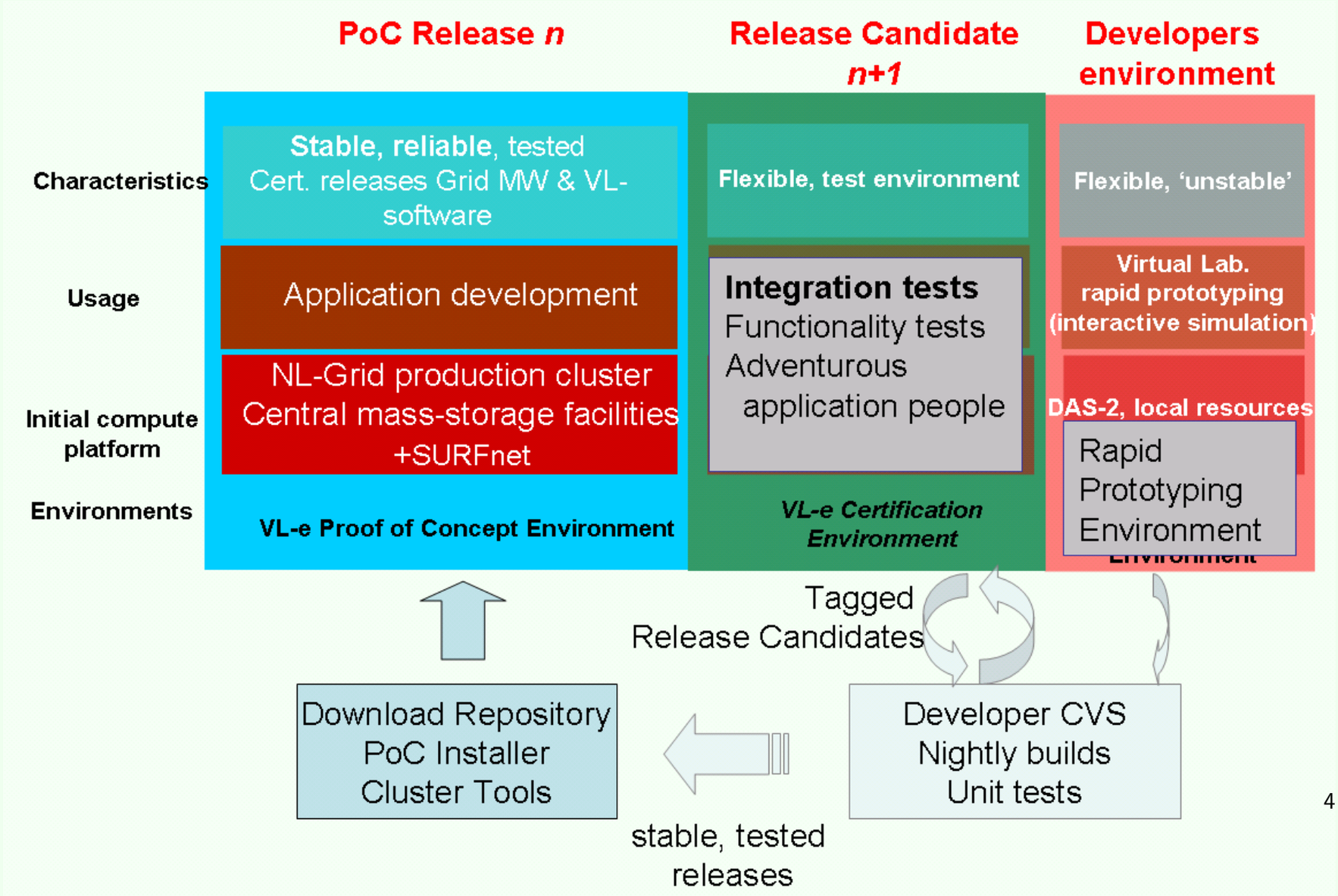
Quote from the http://poc.vl-e.nl/ website:

"The Proof-of-Concept Environment (PoC) is the shared, common environment for e-Science of the Virtual Laboratory for e-Science. In the PoC, the different tools and services used by and provided by the project are available, and bound together in a service oriented approach. The PoC covers three distinct areas:

- *A software Distribution, to be installed by any-one in VL-e interesting in participating in the PoC*
- *A PoC Environment, the ensemble of systems that run the current PoC Distribution*
- *The PoC Central Facilities, those systems running the PoC Distribution that are centrally managed by the Scaling and Validation Programme on behalf of the Project*

The PoC is managed by the VL-e Integration Team (VLeIT). Its mandate defines the composition of the team and the desired PoC architecture."

Intended state of PoC



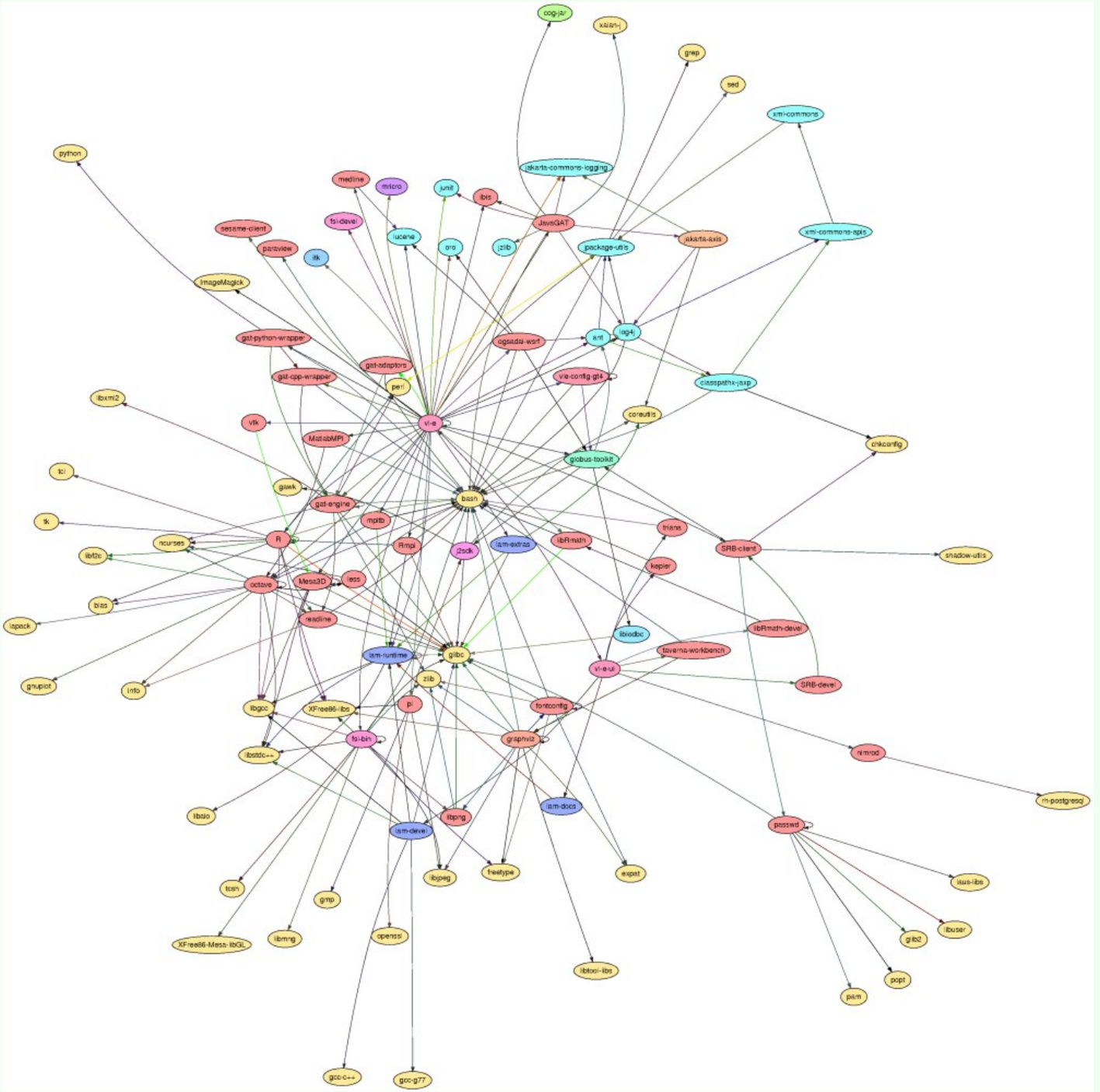
Real state of PoC

- software went in “as-is”,
- no testing unless test suite was available,
- no certification yet.

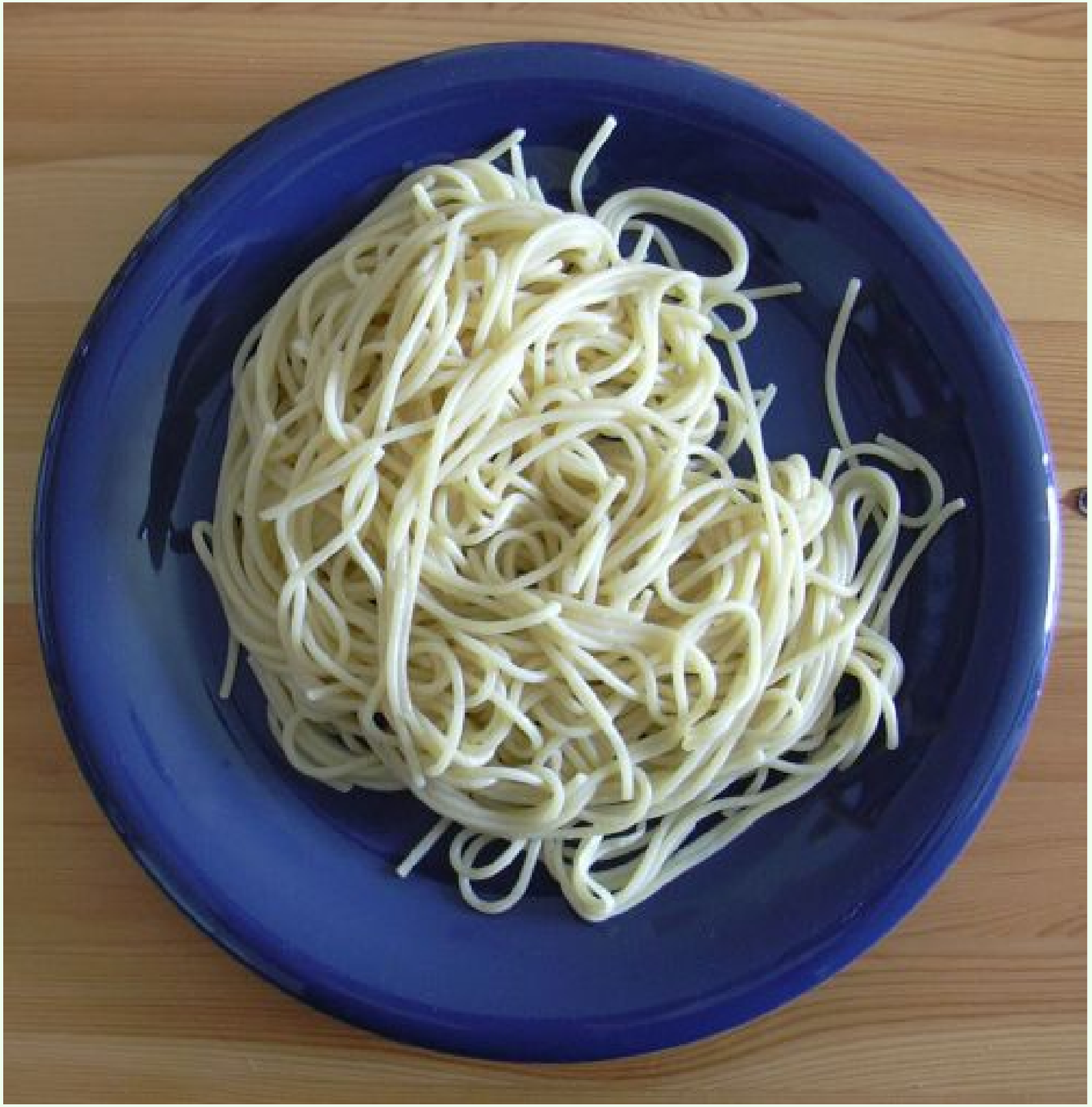
We just needed to pull something off the ground, to raise awareness of the issue.

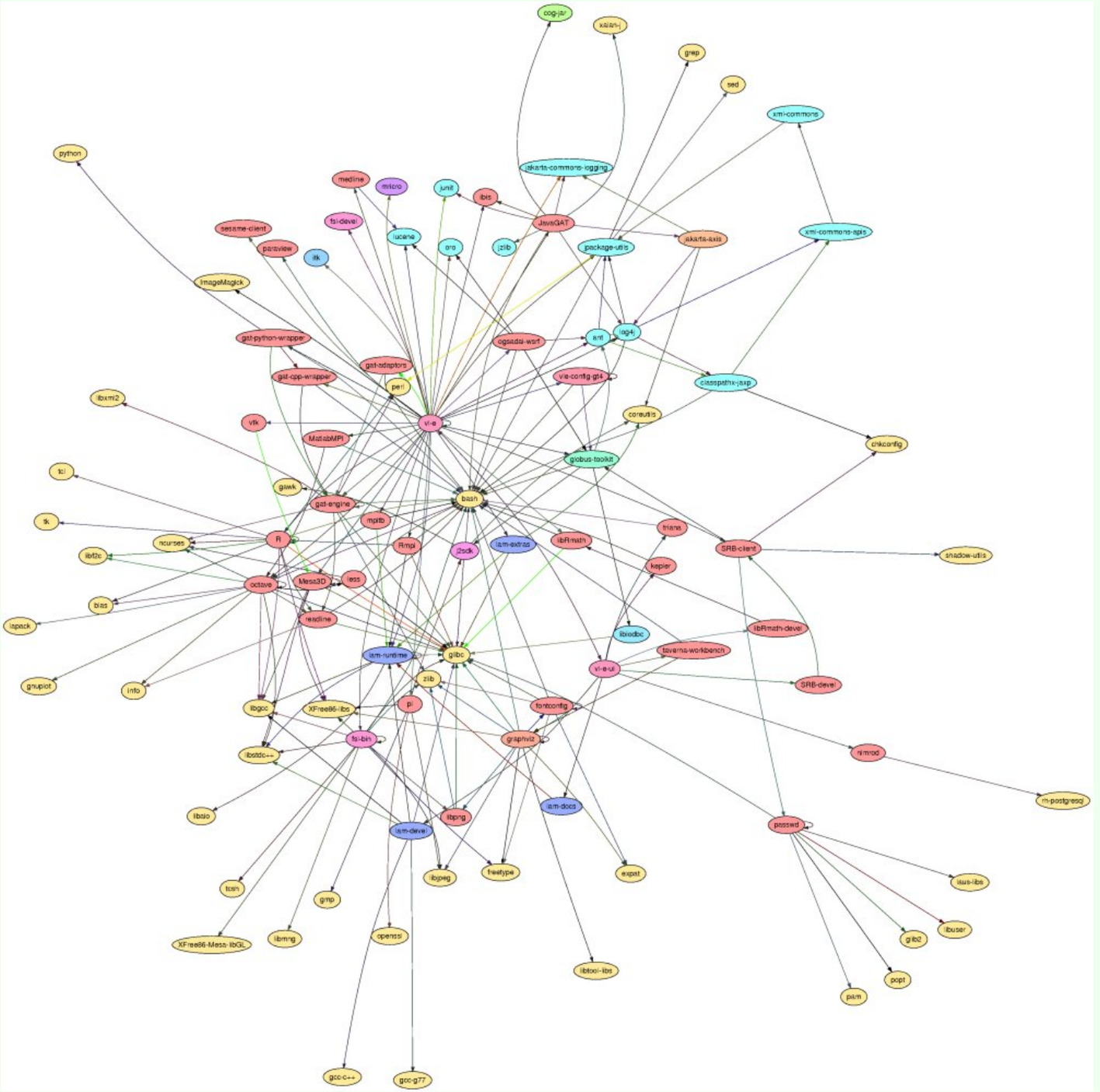
The structure of the PoC

vl·e


cog-jsr
kaleh-j
grep
sed
xmi-commons
python
jakarta-commons-logging
medline
mricro
junit
ibis
JavaGAT
fsi-devel
lucene
oro
zlib
package-utils
akarta-axis
xmi-commons-apis
sesame-client
parsview
itk
ImageMagick
gsl-python-wrapper
ogsadai-wsrf
xml
log4j
gsl-adaptors
gsl-cpp-wrapper
perl
vle-config-gt4
classpathx-jaxp
libxml2
vtk
vl-e
coreutils
MatlabMPI
chkconfig
globus-toolkit
tcl
gawk
bash
gsl-engine
tk
mpitb
triana
R
ncurses
Rmpi
j2sdk
ism-extras
libRmath
SRB-client
shadow-utils
libf2c
kepler
octave
Mesa3D
less
blas
libRmath-devel
lapack
readline
libiodbc
taverna-workbench
ism-runtime
glibc
vl-e-ui
gnuplot
info
zlib
SRB-devel
libgcc
XFree86-libs
pl
fontconfig
fsi-bin
graphviz
nimrod
libstdc++
m-postgresql
libsio
ism-docs
libpng
passwd
ism-devel
freetype
expat
libjpeg
iiuus-libs
tcsh
gmp
libuser
glib2
libmng
openssl
XFree86-Mesa-libGL
libtool-libs
popt
pam
gcc-c++
gcc-g77

vl·e




cog-jsr
kaleh-j
grep
sed
xmi-commons
python
jakarta-commons-logging
medline
mricro
junit
ibis
JavaGAT
fsi-devel
lucene
oro
zlib
package-utils
akarta-axis
xmi-commons-apis
sesame-client
parsview
itk
ImageMagick
gsl-python-wrapper
ogsadai-wsrf
xml
log4j
gsl-adaptors
gsl-cpp-wrapper
perl
vle-config-gt4
classpathx-jaxp
libxml2
vtk
vl-e
coreutils
MatlabMPI
chkconfig
globus-toolkit
tcl
gawk
bash
gsl-engine
tk
mpitb
triana
R
ncurses
Rmpi
j2sdk
ism-extras
libRmath
SRB-client
shadow-utils
libf2c
kepler
octave
Mesa3D
less
blas
libRmath-devel
lapack
readline
libiodbc
taverna-workbench
ism-runtime
glibc
vl-e-ui
gnuplot
info
zlib
SRB-devel
libgcc
XFree86-libs
pl
fontconfig
fsi-bin
graphviz
nimrod
libstdc++
m-postgresql
libsio
ism-docs
libpng
passwd
ism-devel
freetype
expat
libjpeg
iiuus-libs
tcsh
gmp
libuser
glib2
libmng
openssl
XFree86-Mesa-libGL
libtool-libs
popt
pam
gcc-c++
gcc-g77

vl·e

Dependencies for better or for worse

Good:

- know what is required
- automated tools can handle it
- detect conflicts early

Bad:

- a dependency introduces a point of failure
- conflicts are hard to resolve

Keep dependencies down to a minimum!

Java dependencies

Java dependencies are hard to trace.

This could still be useful.

Jar-jar top twelve:

9 × `log4j-1.2.8.jar`	6 × `jaxrpc.jar`
7 × `commons-logging.jar`	6 × `cryptix.jar`
6 × `xalan.jar`	6 × `cryptix-asn1.jar`
6 × `wsdl4j.jar`	6 × `cryptix32.jar`
6 × `saaj.jar`	6 × `commons-discovery.jar`
6 × `puretls.jar`	6 × `axis.jar`

Over one hundred jar files appear at least twice.

Look at JPackage for common Java libraries.

vl·e

Modes of deployment

Gypsy trailer park deployment

No dependencies, self-sustaining, just tug everything along.

Modes of deployment

Gypsy trailer park deployment

No dependencies, self-sustaining, just tug everything along.

Does not scale well!

Modes of deployment

Viral deployment

Unobtrusive, automatic. Minimal needs and assumptions.

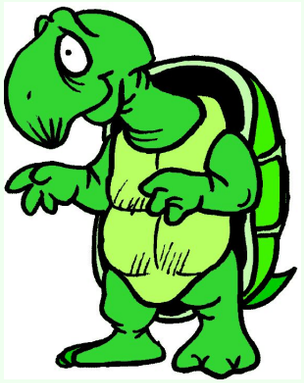
Damage control

Current release is mostly *Aquarium management*. Keep all the fish cohabitating without biting each other's heads off.

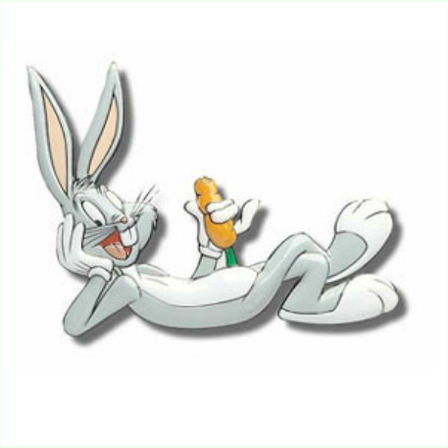
E.g. GT4 vs. GT2; LAM/MPI vs. MPICH.

Towards a stable, collaborative environment

Tortoise mode versus hare mode.



$\Longleftrightarrow$



Every now and again, development has to come out of rabbit prototyping mode and go into TESTudo mode.

Work towards certification, stabilizing. PoC has semi-yearly release cycles.

What's needed?

- Configuration management
- Change management
- Version management
- Package management
- Process management

What's needed?

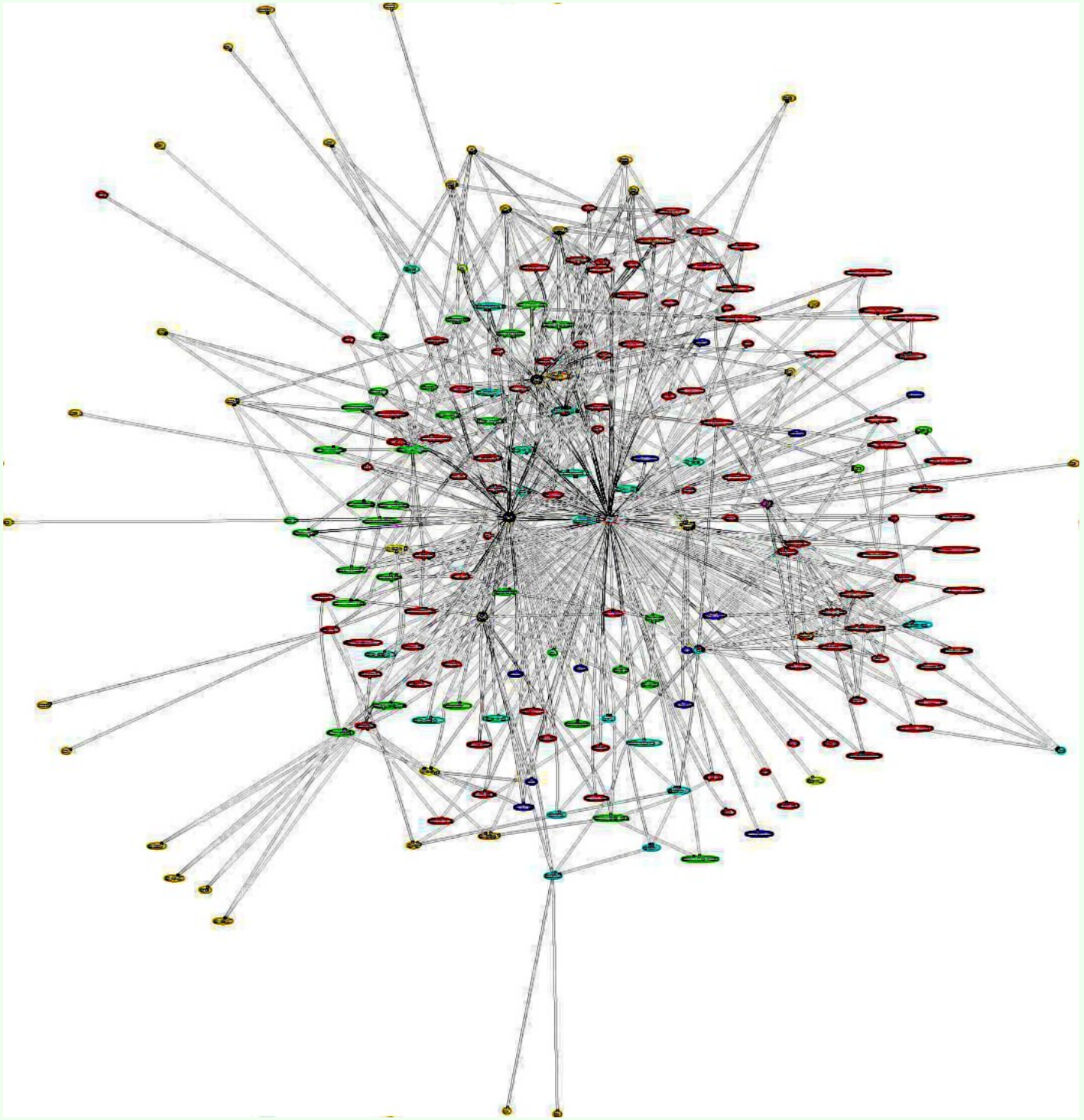
- Provide unit tests
- feature freeze
- cleanup work on the interfaces and APIs

Don't wait until the end of VL-e, this should be an iterative process!

Helpful tools

- VL-e forge: http://gforge.vl-e.nl/
- CVS
- Nightly builds

It could still be worse. . .



vl·e