1 Testing Machine Developer Guideline

1.1 About the project

The Autonomous Province of Bolzano currently offers important digital services (for eg, online applications to start businesses, pay property taxes and initiate e-Payments etc), the number is expected to increase as eGoverment services continue to grow.

The FSCRS project intends to contribute to the creation and improvement of eGovernment services through an innovative process that verifies and tests functionality. The underlying idea of the project is that the accessibility of digital public services is necessary to increase the innovation potential of the region. .

The project takes its name from the Free Software Client Reference System, a specific reference system (OS + defined set of software applications) that during testing simulates a user accessing the services.

1.2 About the software

Testingmachine is used to test eGovernment services. The eGovernment services can vary from country to country but generally it grants citizens access to important documents and information. In most EU countries there are additional services like paying property tax that can be done online. Most governments in Europe are leaning in this direction in order to decrease administrative overhead.

Another challenge to overcome will be testing eGov services on mobile platforms, more specifially automating tests under Android. Good news is this is possible using Selenium. The one major obstacle that we face is getting around the smart card login. So far I am unaware of getting this to work under Android unless you patch the kernel and this is of course no an option. More documentation and research is needed concerning this.

1.3 The eGov Testing Machine

The expected result is the development of a systematic, auto validation process, currently not available on the market, which allows the testing of eGovernment services without the manual intervention of an operator

The eGov Testing Machine can be thought of as a virtual group of people, sitting at the computer and using the eGov services and checking if they work properly, allowing the local Public Administration to test eGov services on a daily basis that are being offered to all citizens.

1.4 Software Overview

The Testing Machine is currently made up by Virtual Machine Manager (tm-vmm) and documentation on how to wite, execute and automate tests of eGov sites in particular but also other softwares.

1.4.1 Virtual Machine Manager (vmm)

tm-vmm is made up by bash scripts that let the user manage various virtual machine software in a general way. See the tm-vmm manual for more information.

1.4.2 eGov Manuals

Writing tests of eGov sites is not hard but we believe that some help will still be useful for most people. The manuals and guides provided together with tm-vmm make it possible to test eGov sites automatically and unattended.

2 Architecture

2.1 Directory overview

2.1.1 bin

Directory containing all programs visible to the user.

- tm-vmm - main program for the management of virtual machines. This script is basically a parser for the functions as stored in the scripts in the scripts directory.
- tm-vmm-auto - outdated script previously used to automate stuff

2.1.2 devel-stuff

A directory containing scripts and other files that can be used while developing Teting Machine. Nothing of this stuff need be included in the distribution.

- bin - contains scripts
  - create-dist.sh - script to create a new distribution

2.1.3 doc

This is the directory conataining the manuals and other documentation. The manuals are written in Markdown format.

- *.md - Source version of the manuals
- generated-files - all files generated from the .md files are placed here
- pics - pictures used in the manuals
- faq-pics - pictures related to the FAQ

2.1.4 examples

Various examples of how to Testing Machine. We will remove this directory since we switched to using a separate repository (tm-examples) for this purpose.

2.1.5 scripts

This is where the functionality of Testing Machine is stored.

- functions
- generate-conf
- vmm-client-android-specific
- vmm-client-qemu-specific
- vmm-client-functions
- loggers - directory containing various scripts that implements different logging functionalities.
  - debug
  - debug-logger
  - txt
  - txt-logger

3 Adding bash code or fixing bugs

3.1 Add a new command line option

If you want to a new command line option to tm-vmm you should update the bin/tm-vmm.tmpl file. Make sure to add code to parse the command line and also as an entry in the help printout.

Update the documentation accordingly.

3.2 Add bash functions

If you want to fix bugs or add new features add this in the scripts directory.

If you add a bash file, make sure to update the Makefile.tmpl accordingly.

4 Writing documentation

All documentation is written in Markdown format. We use pandoc to generate html and pdf.

Each chapter/section is written in separate files to make them more easily reusable. Since there's no *include* functionality in Markdown we had used the Makefile to cp/cat together the various manuals. Perhaps not the most optimal way of doing things - but it works. If you have improvement suggestion you're more than welcome.

If you add a markdown file, make sure to update the doc/Makefile.tmpl accordingly.

5 Building Testing Machine

The scripts and Makefiles in Testing Machine contains variables used to store the installation dir. This is needed to source all bash files.

The scripts and Makefiles are created, with the variables descrbied above updated according to the installation dir, from templates.

5.1 Configure the scripts and Makfile

- Go to the vmm directory

```
cd vmm
```

- Configure the software (you can use –prefix= to specify a non-default installation directory)

```
./configure
```

5.2 Build

- Build the software

```
make
```

5.3 Install and verify

- Install the software

```
sudo make install
```

- Verify the installation (you might need to prepend the path to tm-vmm, if the installation directory is not in your $PATH)

```
tm-vmm --list-clients
```

6 Project Communication

6.1 Reporting bugs

Try to be as precise as possible when reporting bugs. The more information we get the bigger chance we have of fixing the problem.

Use the mailing list below to report bugs.

6.2 Getting involved

How to join: clone the repo, try it out – join the mailing list :)

For more information read the developer guidelines.

6.3 Mailing list

We have one mailing list for the project: `community@testingmachine.eu`

Join this list here: `https://lists.testingmachine.eu/cgi-bin/mailman/listinfo/community`

If you send emails to this list as a non subscriber chances are it will get lost.

If you want to report a bug: * use a github account and add an issue * subscribe to the mailing and send the report to the list

6.4 Blog

https://testingmachine.eu/blog

6.5 Home page

https://testingmachine.eu/

Source code is located here: `https://github.com/tis-innovation-park/vmm`

6.6 Social media

6.6.1 Twitter

https://twitter.com/FSCRS

7 Using git

7.1 Cloning

```
git clone git://github.com/tis-innovation-park/vmm.git
```

7.2 Branch strategy

We don't have any branchinf strategy at the moment.

7.3 Getting write access

If you have patches to the project please send them over, to the community mailing list (community@).

If, after having sent some patches, you feel like you need write access to the repository just contact us and ask.

8 Fixing bugs and issues

When fixing a github issue mark the issue as closed by either:

- adding "closes issue #7" to your git commit message (where 7 is the issue resolved). When you push next time you will see the issue closed.
- Commit and push the fix and log in to github and mark the issue as closed.

9 Discussions

If you need to ask question, need to raise an issue, discuss a bug or simply say something nice, send an email to the mailing list (community@).

Do you wish for another media (IRC, forum...) propose it on the mailing list.

10 Generating Manuals

To generate manuals you simply enter the doc directory:

```
cd doc
```

and then type:

```
make
```

The generated documentation are stored in the generated-files directory.

10.1 Requirements

To generate the manuals from the Markdown sources you need to install:

- pandoc
- pdflatex

We provide a list of packages to install for some GNU/Linux distributions:

10.1.1 Debian based GNU/Linux distributions

```
sudo apt-get install pandoc texlive-latex-base
```

11 Releasing the software

11.1 Manual release

To release we go through the following procedure:

- Edit the version number in the configure script

```
emacs -nw configure
```

- Commit all your code and push it to github

```
git commit -m <some comment>
```

- configure and build

```
./configure --prefix /tmp && make clean all install
```

- Update the tm-manuals repository `cd doc && make clean all release-doc; cd ..`
- Verify the build
  - Check the version `/tmp/bin/tm-vmm --version`
  - Make sure all script are installed `/tmp/bin/tm-vmm --list-clients`
- Tag the code with the same version as you used in the configure script.
  `git tag -a <version number> -m '<version number'`
- Push the code and the tag `git push` `git push --tags`
- Build
  `make dist`
- Upload `tmp/testing-machine-0.14.tar.gz` to tm-releases

11.2 Scripted release

We've written a script to help release a new version of Testing Machine. Before you call it:

- Edit the version number in the configure script

```
emacs -nw configure
```

- Commit all your code and push it to github

```
git commit -m <some comment>
```

Now we're ready to call the script:

```
./devel-stuff/bin/create-dist.sh --version <new-version> --force-tag
--push
```

The scripts accepts some options:

- `--version <VERSION>` - Set the version number for the new release
- `--push` - Push the code (and tags) up to github
- `--force-tag` - If the tag already exists, overwrite it

12 Releasing the manuals

12.1 Scripted release

```
cd doc
```

```
make clean all release-doc
```