E-Learning – VELS
Usermanual

Project: E-Learning – VELS
Author(s): Martin Mosbeck, Andreas Platschek, Gilbert Markum, LingYin Huang
Reviewer: Axel Jantsch
Version: 1.3
Date: November 5, 2018

Contents

Todo list

Version	Autor	Date	Comment
0.1	Martin Mosbeck	14.09.2015	First shot at User Manual.
0.2	Martin Mosbeck	28.05.2016	Updates for public release.
0.3	Martin Mosbeck	04.10.16	Updates about configurability.
0.4	Martin Mosbeck Gilbert Markum	04.10.16	Added Xilinx ISE ISim Install Notes
0.5	Martin Mosbeck	27.05.17	Rework, incorporation of changes to config file and system, incorporation of the specification
0.6	Gilbert Markum Martin Mosbeck	17.07.17	Added section “Creating a new task for the VELS system”
0.7	LingYin Huang	31.07.17	Updates Debian 9 (Stretch) system prerequisites and minor typo correction in section "System Setup"
0.8	Martin Mosbeck	20.09.17	Inserted documentation for multi-language feature
0.9	Martin Mosbeck	02.01.18	Corrected typos, adding additional information to make documentation easier to understand, section on task queue modes
1.0	Martin Mosbeck	09.07.18	Deleted information about deleted skipping mode
1.1	Martin Mosbeck	09.07.18	Added information about TaskErrorNoticd message type
1.2	Martin Mosbeck	01.11.18	Restructuring of Prerequisites and adding QuestaSim notes.
1.3	Martin Mosbeck	03.11.18	Changes due to rename of CommonFile to BackendInferfaceFile.

1. Manual Overview

The following document aims to inform an operator on on how the autosub system works and how to configure and run a course using the autosub system. This manual is written primarily for using autosub with VELS (VHDL E-Learning System). Nevertheless the autosub system and the described structures are designed to be usable for diverse E-Learning purposes.

The first part of this manual (Section 2) outlines the interaction and use cases for a user and an operator with the E-Learning system.

The second part of this manual describes VELS, which itself can roughly be divided into 3 parts:

1. The autosub submission system (see Section 3)
2. The configuration and status web interface VELS_WEB (see Section 4)
3. The tasks interface (see Section 5)

The third part of this manual depicts prequesites (see Section 6), notes on supported testing backends (see Section 7) and which steps have to be taken for system setup (see Section 8).

The fourth part describes how to create new tasks for VELS (see Section 9).

The Appendix (Section 10) list implementational details of VELS.

2. System Interaction

The VELS system is an E-Learning system for students who are learning the Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL). The interaction between students and the system is solely via E-Mail. Configuration of the system is done with a configuration file (for configuration items that can not change during runtime) and a web interface (for configuration items that can change dynamically).

The VELS E-Learning system has 2 distinctive user groups: course operators and students. To satisfy the use cases for each of the 2 user groups the following interfaces have been defined:

- VELS E-Mail Interface
- VELS Web Interface
- VELS Direct Server Access

The different use cases for students and operator and the responsible interface can be seen in Table 1 and Table 2.

Use Case	Responsible interface
register with the system	VELS E-Mail Interface
get status in course	VELS E-Mail Interface
request/get a task	VELS E-Mail Interface
submit a task solution	VELS E-Mail Interface
ask a question	VELS E-Mail Interface

Table 1: Use cases for students

Use Case	Responsible interface
configure a course	VELS Web Interface
configure a task queue	VELS Web Interface
modify task generation or testbench generation	VELS Direct Server Access
create a new task	VELS Direct Server Access
view the progress for students	VELS Web Interface
view task statistics	VELS Web Interface
read log files	VELS Direct Server Access

Table 2: Use cases for course operators

2.1. VELS E-Mail Interface

The VELS E-Mail Interface is the primary interface for students. Students are uniquely identified by their E-Mail address to interact with this interface. The student has to interact with a single, non-changing E-Mail address during the whole duration of a course. These E-Mail addresses have to be added to the system's whitelist by a course operator in order to enable them to be authorized for interaction. If a user that is not on the whitelist sends an E-Mail to the system, the system replies with a message to use the whitelisted address or contact a course operator.

The different actions a student can take are defined via E-Mail subject and address. The following cases when using a whitelisted E-Mail address and system reactions are implemented:

- *Arbitrary Subject, user not registered:* Registration is initiated. Registration with the system creates the appropriate data entries in the database for the student. If adding the new user was successful, the user gets an welcome E-Mail with general information on how the system works.
- *Subject contains the word "Question":* The student has a question about the course, this question is forwarded to all course operators. The student receives a confirmation that the question has been passed on.
- *Subject contains the phrase "Question Task N" where N is the task number:* The student has a question about a specific task of the course, this question is forwarded to all task operators for task N. The student receives a confirmation that the question has been passed on.
- *Subject contains the phrase "Result Task N" where N is the task number:* The student wants to hand in a solution, for Task N. Students are allowed to hand in solutions for all tasks that have already been solved previously, as well as the one task that has not been solved yet – they are not allowed to hand in solutions for tasks they did not receive the task description yet. The solution is tested on the server. If the task solution was appropriate the student will receive the next task (or a congratulations message, if no more examples are available). If the tests were not successful, the student receives an error message that gives a hint on what may be wrong (output of the simulation tool, expected output vs. actual output, etc.)
- *Subject contains the word "Status":* The student is send an E-Mail with the list of his completed tasks and his current task description.
- *default:* The student is sent an E-Mail with the usage for this interface.

Depending on the chosen task queue mode additional keywords are accepted (for a closer description of different modes look at Section 8.3):

With request mode:

- *Subject contains the word "Request Task N":* The student wants to request the task N. If the task can be given to the student at this moment, the student will receive an E-Mail containing all informations on the task.

2.2. VELS Web Interface

The VELS Web Interface is a configuration and status tool for the course operators.

Course operators can create a course. A course consists of a task queue which is a selection of the available tasks. The following actions for course configuration are implemented:

- Configuration of the task queue
- Configure course registration deadline
- Modify course registration E-Mail whitelist
- Modify archive directory
- Set administrator E-Mail

Course operators can view the progress of each student and is presented with the following informations:

- Student name.
- Task the student is working on and the tasks the student has completed.
- Score which was accumulated by the student.
- Number of submissions for each task the student has received.
- List of attached files for each task the student has received.

Course operators can view statistical informations about their course:

- Number of wrong and right submissions for each task.
- General statistical informations about the course (e.g number E-Mails received/sent).

The VELS Web Interface may also be used by the course operator to read the results at the end of the semester, that is the points that have been scored by the individual students. In depth description on how to use the VELS_WEB interface can be found in Section 4 and Section 8.

2.3. VELS Direct Server Access

The VELS Direct Server Access is used by course operators to modify code for task generation, description and testing of existing tasks, read log files or fix bugs of the VELS system. It also can be used to view every submission a student has done.

3. Autosub Submission System

The autosub submission system has the following responsibilities:

- Fetch, process, send and archive student E-Mails.
- Store and manage tasks, users and submissions.
- Initiate generation of tasks and testing of submissions.
- Produce statistics.

In the following *Structured Analysis* will be used to specify the system. If you are not familiar with this method, you can find details e.g. in [DeM81, Goo01, Coo03].

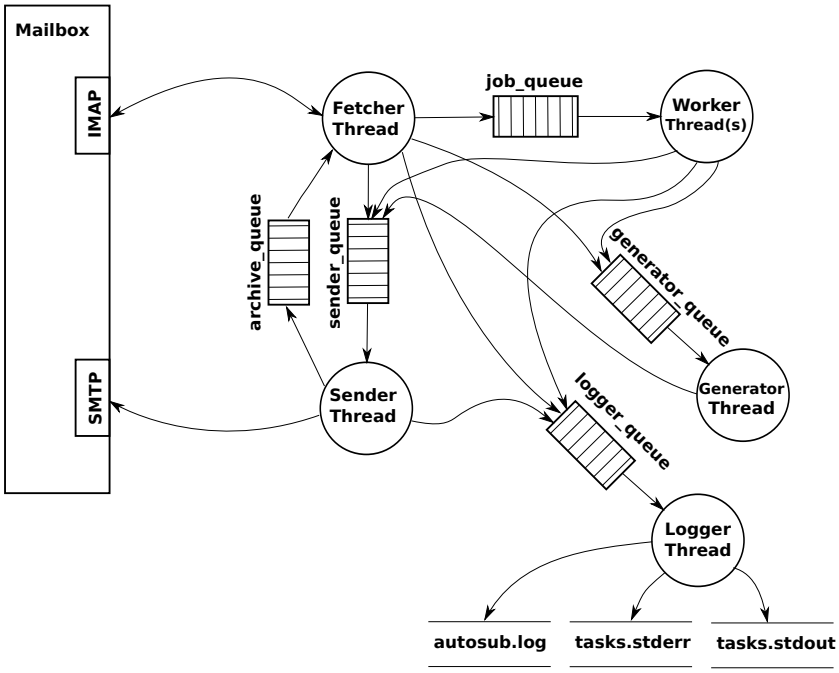

Figure 1: DFD0 – High-Level Data-Flows in the Submission System.

The autosub submission system is a daemon implemented in Python3 that is composed of multiple threads communicating over queues. This multi-threading approach is especially important for the testing process. As the test of a single task can take rather long and each student can hand in (correct and incorrect) solutions. Furthermore the goal is to assure, that there still is progress, even if a test is running. A view of the parts and data-flows of the autosub system can be seen in Figure 1. The individual entities are described in the next subsection, the message queues in the Appendix Section 10.2.

3.1. Description of Entities

MB Mailbox

Description	Mailbox to interface with students.
Rate	Asynchronously by students, periodically with configurable interval by the E-Learning system.
Comment	The Mailbox is used to interface with the students. The Mailbox is accessed via Internet Message Access Protocol (IMAP) for reading new E-Mails and Simple Mail Transfer Protocol (SMTP) to send E-Mails. After the action triggered by an received E-Mail has been processed, the E-Mail is archived into a configurable folder on the IMAP server.

1 Fetcher thread

Description	Fetch E-Mails from the Mailbox.
Rate	Periodically with configurable interval.
Comment	The Fetcher thread periodically checks the Mailbox for new E-Mails. This thread initiates actions based on the user E-mail address and the subject. Further information concerning the E-Mail interface can be be seen in Section 2.1

2 Generator thread

Description	Generate a (unique) example for a user.
Rate	Asynchronously triggered by the E-Learning System when a new task for a user has to be created.
Comment	The Generator thread initiates generation of unique tasks for students. The Generator thread is blocking on the generator_queue, until the generation of a new example is requested. As soon as the example was generated, a request to send it out is put into the sender_queue.

3 Worker thread(s)

Description	Test the task submissions of students.
Rate	Asynchronously triggered upon arrival of a new task submission by a student.
Comment	The Worker threads are the ones which do the initiate testing submissions by a student. If the Fetcher thread receives an E-Mail with a result submission, a new message is added to the job_queue. The worker threads are using a blocking read on the job_queue to wait for work, as soon as the new message is added by the Fetcher thread one of the workers will read it and start the tests for the users submission.

4 Logger thread

Description	Format log messages and store them in the log file.
Rate	Asynchronously triggered by all other threads.
Comment	In order to enforce a common logging format, and allow the setting of a common log-level, all threads just put their log-messages into the logger_queue. The Logger thread decides (depending on the log-level and configured log-threshhold) whether or not the message shall be logged or not. If so, the message is formatted and written to the log file. The log-level threshold can be configured in the config file. The following log-levels are implemented (from lowest to highest): **DEBUG:** Print all kind of information that might be helpful to debug problems. This includes, what user sent which E-Mail, what action was taken, what was the result of that action. **INFO:** Information that might be of interest, even if not in debugging mode. **WARNING:** A problem occurred, but it did not lead to a long-term problem (e.g. could not connect to the mailserver, but after retrying it worked). **ERROR:** A problem that lead to a long-term malfunction of the submission system occurred, the system will not be able to recover from this problem.

5 Sender thread

Description	Sends E-Mails to the students
Rate	Asynchronously by Fetcher, Worker or Generator thread.
Comment	If a message has to be returned to the student, the thread that wants to send the message just puts the necessary information into the sender_queue. The sender thread then takes that information, formats an E-Mail and sends it out using SMTP. The unique message-id is passed from the fetcher (possibly via other threads) to the sender thread. If the sender thread has sent out the answering E-Mails, the E-Mail with the given message-id is moved into an archive folder on the mail server.

6 Dailystats thread

Description	Creates statistical information.
Rate	Activated every 12 hours.
Comment	The dailystats thread is used to get a timeline of some very basic statistics. Currently the number users, sent E-Mails, received E-Mails and received questions over time are evaluated and plotted into a graph. The graphs are stored and can e.g. be viewed in the Statistics tab of the VELS web interface.

3.2. Datastores

For storing data two databases are used. The name and location of those databases can be configured through the configuration file, in the following we will refer to them with the default name that is used, if no other name is provided in the configuration file:

semester.db: The `semester.db` database is used to store all data that is specific to one semester. This includes, users, parameterization of tasks assigned for users, statistics, etc. The `semester.db` default location is in `"autosub/src"`.

course.db: The `course.db` database consists of configuration data that will very likely be reused in multiple semesters. This separation makes it very easy to start a new semester: just backup `semester.db` to some safe location and remove it in the original location (a new empty one will be made automatically upon starting the autosub daemon). Then use the VELS web interface to perform the small modifcations(year/semester,deadlines, etc.) needed in `course.db`. The `course.db` default location is in `"autosub/src"`.

Specifics about the tables of these databases can be found in the Appendix Section 10.3 and 10.4.

In addition there is one important directory that is used by autosub to store data in, the directory: `autosub/src/users`. In this directory a unique directory for every user is created, and this directory the description of each individual received task and submissions for the individual tasks are stored. Every directory has the form `"autosub/src/users/<user_id>/Task<nr>/"`. Submissions are stored in separate folders which are named `"Submission<nr>_<date>_<time>/"`. Task description files are stored in the folder named `"desc/"`.

3.3. Config File

The autosub system uses a config file (.cfg) which is passed to the daemon to configure the individual threads and overall system. The file is read when starting the daemon. If no

.cfg file is specified, the used file is default.cfg located in `"autosub/src/"`. Information about the config file can be found in the Appendix Section 10.1 and an example config file in in Section 8.4.

3.4. Logging and error detection

The autosub system is implemented as a daemon process. As such it does not directly print any information to the `stdout` (i.e. the screen), instead all messages for the administrator are stored in log files. All log and error files are stored in a configurable directory (default `"autosub/src"`).

autosub.log This is the main log file that is used by the actual autosub daemon software to log everything. The format in this log file is as follows:

```
<date> <time> [<logger name>] <loglevel> <logmessage>
```

Here is a small section of `autosub.log` taken from startup – first the daemon checks whether a directory `users` used to store the users submissions is existing, then it checks if necessary tables in the database exist. After those checks have been done further threads (described in Section 3.1) are started, and announce themselves.

```
2015-09-17 22:23:40,629 [autosub.py  ] WARNING: Directory already exists: users
2015-09-17 22:23:40,630 [autosub.py  ] DEBUG: table SpecialMessages does not exist
2015-09-17 22:23:43,679 [autosub.py  ] DEBUG: table TaskConfiguration does not exist
2015-09-17 22:23:43,955 [autosub.py  ] DEBUG: table GeneralConfig does not exist
2015-09-17 22:23:45,878 [sender      ] INFO: Starting Mail Sender Thread!
2015-09-17 22:23:45,884 [fetcher     ] INFO: Starting Mail Fetcher Thread!
2015-09-17 22:23:45,884 [fetcher     ] DEBUG: Imapserver: 'mail.intern.tuwien.ac.at'
2015-09-17 22:23:45,884 [generator   ] INFO: Task Generator thread started
2015-09-17 22:23:45,890 [activator   ] INFO: Starting activator
2015-09-17 22:23:45,890 [Main        ] INFO: Used config-file: testzid.cfg
2015-09-17 22:23:45,891 [Worker1     ] INFO: Starting Worker1
2015-09-17 22:23:45,892 [Main        ] INFO: All threads started successfully
2015-09-17 22:23:45,892 [Worker1     ] INFO: Worker1: waiting for a new job.
2015-09-17 22:23:45,893 [Worker2     ] INFO: Starting Worker2
```

As you can see from theses examples, a lot of information is available, with exact timestamps and a way to know which thread the message is coming from. In addition the debug level gives a sense of how important (or even critical) the message is. The log-level threshhold can be configured using the config file.

Autosub tries to give information on severe problems on all available channels. An example would be, if no task is configured, the email sent to the student will contain the message that something went wrong as well – although the error can be seen in the `autosub.log` file as well.

4. The web inteface VELS_WEB

The configuration and status web interface VELS_WEB was designed as an easy interface for a course operator to view and change course parameters and monitor the course progress. It is implemented with web2py and can be run as a daemon accessing the datastores of the autosub submission system. This web interface was only designed for internal operator use! Therefore it should only be reachable via the internal networks using a web browser.

The menu items of the VELS_WEB are the following:

Start: The start page.

Users: View of the registered students, allows to change or a student via the "Edit" button, delete a student via the "Delete" button and show the progress of a student in the course via the "View" button. The table is sortable by column by clicking on the head of the column.

Tasks: The task configuration, will be discussed in Section 8.9.

Whitelist: Change whitelisting for students, will be discussed in Section 8.8.

General Config: Change dynamic configuration items of VELS, will be discussed in Section 8.7.

Statistics: View statistics about sent and received emails and task success of the students.

User Tasks: View mappings from students to individual tasks.

Usage of the VELS_WEB interface will be discussed with system setup in Section 8.

5. The Tasks System

5.1. Minimal and optional task structure

Every task is located in an individual folder in the tasks directory (configurable via the autosub config file). Certain structures with minimum behavior are needed in order to use the task with the autosub system. In detail instructions how to create a taks can be found in Section 9.

Minimal	• **Generator executable:** Generates random task parameters for the task, creates entity and behavioral vhdl files and description pdf file for the student. Copies these files to the user taskpath (`"autosub/src/users/<user_id>/Task<nr>/"`). Declares which files shall be attached to the task description E-Mail and the task parametes and adds them to the autosub system using the tool `"autosub/src/tools/add_to_usertasks.py"`. • **Tester executable:** Given the individual task parameters, tries to test the student's solution located in `"autosub/src/users/<user_id>/Task<nr>/"`. Generates an individual testbench for the student's solution. Tests the solution and creates feedback textfile `"error_msg"`, error attachments in the directory `"error_attachments"` and tells the autosub system the test result via returncode • **description.txt:** Contains a textual task description. The text in description.txt is sent to the user as the body of the E-Mail that contains a new task description.

Certain optional structures have proven to be useful.

Optional	• **Directory scripts:** Scripts that are called from the executables in order to aid the generation or test process • **Directory templates:** Files that are filled with parameters and can be used to generate entity declarations, testbenches, description files. • **Directory static:** Files that are static for the task, therefore the same for every student. • **Directory exam:** Files that are needed for the VELS exam mode.

An usual sequence for the task generation in VELS is:

1. The generator executable is called by the autosub Generator thread.
2. The generator executable calls a task generation script `"scripts/generateTask.py"`.
3. The task generation script generates random task parameters and returns them, fills a LaTeX description template and vhdl templates and stores the results in the directory `"tmp/"`.
4. The generator executable generates the task description pdf file from the filled template and copies it, all filled vhdl files and static vhdl files to the users task description path `"autosub/src/users/<user_id>/Task<task_nr>/desc/"`.
5. The generator executable calls `"autosub/src/tools/add_to_usertasks.py"` with the task parameters and path to files that belong to the individual student task. This tool saves these informations per user in the semester database for creation of the student task E-mail.

An usual sequence for submission testing in VELS is:

1. The tester executable is called by a autosub Worker thread.
2. The tester executable calls a testbench generation script `"scripts/generateTestBench.py"`.
3. The tesbench generation script creates a testbench with test vectors and returns it.

4. The tester executable stores the testbench in the user task directory `"autosub/src/users/<user_id>/Task<task_nr>"` and copies needed files from the user task description path `"autosub/src/users/<user_id>/Task<task_nr>/desc"` to the user task directory `"autosub/src/users/<user_id>/Task<task_nr>"`.
5. The tester executable checks if the needed submission files are present in the user task directory. These files are stored from E-Mail attachments by the Fetcher thread. If they are not present an error is written to a `error_msg` file and the process returns failure. This initiates the sending of an E-Mail to the student.
6. The tester executable analyzes files generated by the task system, if errors occur the process stops stops and returns a task error. This initiates the sending of an E-Mail to the student and system administrators.
7. The tester executable analyzes the student submission files, if errors occur they are written to a `error_msg` file and the process stops and returns failure. This initiates the sending of an E-Mail to the student.
8. The tester elaborates and runs the testbench, if the tests fail meaningful messages are written to a `error_msg` file, the process stops and returns failure. If the test was successfull the `error_msg` file remains emty and the tester returns success. Both cases initiate the sending of an E-Mail to the student.

5.2. Tasks error detection and logging

The autosub daemon calls generator and tester scripts for the individual tasks. The logging of these scripts is done by piping the output of called generator and tester scripts into seperate log files. All output to stdout is piped into the logfile `tasks.stdout`, allo error output to stderr is piped into the logfile `tasks.stderr`. The location of these files (default `"autosub/src"`) can be configured using the log file. Messages to the files are logged in the format:

```
--------------------------------------------------------------------
<date> <time> [Task<task_nr>(<worker_nr)] <ERROR/INFO>:
--------------------------------------------------------------------
<logmessage>
```

An example for the tester of a submission from the gates task to `tasks.stdout` is:

```
--------------------------------------------------------------------------------
2017-05-09 15:07:16,124 [Tester1(10) ]INFO:
--------------------------------------------------------------------------------
Parsing VHDL file "gates.vhdl" into library work
Parsing VHDL file "gates_tb_10_Task1.vhdl" into library work
Parsing VHDL file "IEEE_1164_Gates_pkg.vhdl" into library work
Parsing VHDL file "IEEE_1164_Gates.vhdl" into library work
Parsing VHDL file "IEEE_1164_Gates_beh.vhdl" into library work
Parsing VHDL file "gates_beh.vhdl" into library work
Task 1 analyze success for user 10!
Task 1 not using the provided gate entities for user 10!
```

6. Autosub & Tasks Prerequisites

In the following, the installation of pre-requisites for autosub and the tasks are documented for Debian. The system might work with other distributions, if you find the packages that have to be installed, please provide us with instructions so we can expand this section.

6.1. Debian 7 (Wheezy)

Installs from the apt package pool:

```
sudo apt-get install python3 python3-mock python3-pip python3-nose
sudo apt-get install zip flip git gnat-4.6-base libgnat-4.6
sudo apt-get install texlive-latex-extra pgf graphviz build-dep
sudo apt-get install zlib1g-dev rsync
```

Next we need to upgrade easy_install for python3 to a newer version – this is the prerequisite for the installation of the matplotlib for python3.

```
easy_install3 -U distribute
```

The following packages need to be installed using pip:

```
pip-3.2 install pyqt5
pip-3.2 install matplotlib
pip-3.2 install bitstring
pip-3.2 install graphviz
pip-3.2 install jinja2
```

6.2. Debian 8 (Jessie)

Installs from the apt package pool:

```
sudo apt-get install python3  python3-mock python3-pip python3-nose
sudo apt-get install zip flip git gnat-4.9-base libgnat-4.9
sudo apt-get install texlive-latex-extra pgf graphviz  build-dep
sudo apt-get install zlib1g-dev python3-pyqt5 python3-matplotlib rsync
```

The following packages need to be installed using pip:

```
pip3 install bitstring
pip3 install graphviz
pip3 install jinja2
```

6.3. Debian 9 (Stretch)

Installs from the apt package pool:

```
sudo apt-get install python3 python3-mock python3-pip python3-nose
sudo apt-get install zip flip git gnat-6 libgnat-6
sudo apt-get install texlive-latex-extra pgf graphviz build-essentials
sudo apt-get install zlib1g-dev python3-pyqt5 python3-matplotlib rsync
```

The following packages need to be installed using pip:

```
pip3 install bitstring
pip3 install graphviz
pip3 install jinja2
```

6.4. Patches to be applied

Unfortunately there is a small bug in one of the latex packages that we use (pst-circ), therefore you need to apply a patch to that package:

```
cd /usr/share/texlive/texmf-dist/tex/generic/pst-circ
patch < /path/to/autosub/doc/patch/pst-circ.tex.patch
```

In case logic gates in the pdfs you are not beautiful as you are used to, it might have happended that an update was applied over that patch and you need to re-apply it.

7. Notes on supported Backends

VELS supports multiple backends for the testers. In this section you find install or configuration notes concerning them.

7.1. GHDL

GHDL's current official releases are located at https://github.com/ghdl/ghdl/releases

For Debian 7 we used the version from sourceforge: http://sourceforge.net/projects/ghdl-updates/files/Builds/ghdl-0.31/Debian/ and installed it using dpkg:

```
dpkg -i Downloads/ghdl_0.31-2wheezy1_amd64.deb
```

For Debian 8 we used the version from sourceforge: http://sourceforge.net/projects/ghdl-updates/files/Builds/ghdl-0.33/debian/ and installed it using dpkg:

```
dpkg -i Downloads/ghdl_0.31-2wheezy1_amd64.deb
```

Starting with Debian 9 the github releases seem to be the way to go: https://github.com/ghdl/ghdl/releases

7.2. Xilinx ISE 14.7

The ISE ("Full Installer for Linux") can be downloaded from Xilinx's official download page. It requires registration and licensing agreement, but there is no charge.

For installing ISE to the default location /opt/Xilinx/ where the VELS system expects it to be, you need permission to write to this location. So we will use the user root to call the graphical installer. To do so we need to allow root to use the users X server. As a user run:

```
$ xhost +
```

to allow root (or anyone) to connect to your users X server temporarily. Now as root start the graphical installer, located at the top of the extracted download, with

```
# ./xsetup
```

In case that you are using the KDE desktop environment, you have to remove the `QT_PLUGIN_PATH` environment variable before starting the graphical installer:

```
# unset QT_PLUGIN_PATH
```

If you are running the VELS system on a machine without a graphical user interface, the best way we found to get Xilinx's ISE onto a remote debian server was to copy the entire installation directory after it has been installed on a non-headless machine.

7.3. Mentor QuestaSim/ModelSim

QuestaSim is a version of ModelSim, therefore the following notes also apply to ModelSim.

Although QuestaSim is officially only available for RedHat, it works fine with Debian. After install you have to create an setup file and specify it in the common tester. This setup file should export the path to the binaries and set the license environment variable. It could look like this:

```
QUESTASIM_BIN=/eda/questasim/linux_x86_64
export PATH="${PATH}:${QUESTASIM_BIN}"
export MGLS_LICENSE_FILE=...
```

Listing 1: QuestaSimSetup.sh

8. Setting up the VELS system

8.1. Email Configuration

VELS has to be connected to an E-Mail account from which it can fetch E-Mails and send E-Mails. As VELS itself does not include an E-Mail server, you will have to set up a E-Mail server or create an account on an existing server. The account has to be reachable via SMTP and IMAP. Note possible secure connection modes (ssl/starttls), ports and connection information of your account/server as you will need them when you create the VELS config file (see Section 8.4).

Additionally the account needs to have a folder to archive processed E-Mails, VELS by default expects it to be named archive (can be changed in VELS_WEB → General Config). This way only non fully processed E-Mails will be in the inbox of the account. If VELS crashes you can set them to unread and the VELS system will process them.

Make sure the account you choose has an empty inbox when starting the autosub daemon for the first time, as VELS will try to process all unread E-Mails, even if no user has been whitelisted yet!

8.2. Getting the VELS system

The first thing to do when you want to install the autosub submission system and the VELS web interface is to clone the git repository:

```
git clone https://github.com/autosub-team/autosub.git
```

To fetch the latest version of the available tasks, run the script update_tasks.sh in the autosub root folder.

```
./update_tasks.sh
```

This will clone the task repositories specified in the script and move them to the folder tasks/implementation/VHDL .

To fetch a newer version of the tasks at any time you can rerun the update_tasks script with the parameter refetch. Warning: this will delete all the content inside the tasks folder (tasks/implementation/VHDL) and then fetch the new tasks.

```
./update_tasks.sh refetch
```

If you just want to add additions done to the tasks without cleansing of the folder first, run the update_tasks script without the parameter refetch.

```
./update_tasks.sh
```

8.3. Chosing the task queue mode

VELS can be used in two distinct task queue modes: request mode and linear mode.

Request mode
In this mode students request a task out of the configured tasks by request. This enables students to choose which tasks they want to solve. In order to choose this queue mode allow_requests has to be set to once or multiple in the configuration file (see Section 8.4, Section 10.1).

Example scenario with allow_requests = multiple
To be added

Example scenario with allow_requests = once
To be added

Linear mode
In this mode students get a linear queue of tasks. In order to choose this queue mode allow_requests has to be set to no in the configuration file (see Section 8.4, Section 10.1). In linear mode the student get sent the first task at registration with the system (if the task is active at that moment).

Example scenario for linear mode
To be added

This mode can be combined with auto_advance, which auto advances users to a task once it is becoming acvive by its TaskStart (see Section 8.4, Section 10.1).

Example scenario with auto_advance = yes
To be added

8.4. Configuration File Creation

The autosub config file is used to configure the connection to the mail server and set configuration parameters for the system. In order to run the autosub daemon a config file has to be created. The config file fields each belong to one of the groups embraced in [...]. An example config file can be seen below, the full list of fields and their meaning is described in Appendix Section 10.1.

```
[imapserver]
servername: imap.gmail.com
serverport: 993
security: ssl
username: submission@gmail.com
password: mysupersecurepassword
email: submission@gmail.com

[smtpserver]
servername: smtp.gmail.com
serverport: 587
security: starttls
username: submission@gmail.com
password: mysupersecurepassword
email: submission@gmail.com

[system]
num_workers: 20
queue_size: 200
poll_period: 5
log_dir: /home/martin/autosub/src
log_threshhold: INFO

[course]
tasks_dir: /home/martin/autosub/tasks/implementation/VHDL
course_name: My Cool Course
mode: normal
auto_advance: no
allow_requests: no
```

Listing 2: example.cfg

8.5. Starting the Daemon

Autosub is implemented as a daemon process, that means all messages provided are written to files (see Section 3.4 and 5.2 for details on those files) – nothing is written to the console. The daemon is started using a shell-script located in `autosub/src/`:

```
./autosub.sh start
```

This starts the daemon using the default configuration file named `default.cfg`. If you wan to use your own configuration file, you have to pass the files name to the script when starting the daemon:

```
./autosub.sh start myconfig.cfg
```

To stop the daemon just run the command:

```
./autosub.sh stop
```

8.6. Setting up the VELS_WEB Configuration Interface

To use the VELS_WEB Configuration Interface it has first to be installed and configured using these steps:

Change to the VELS_WEB directory.

```
python3 installer.py <pathtoautosub> <pathtoconfigfile>
```

Use the same configfile you used for starting the autosub daemon! This step will also download web2py to the user's home folder and set needed symbolic links to connect VELS_WEB to the autosub system.

If you change parameters in the category [system] in your autosub config file or need to switch to another config file run the installer with the reinstall flag:

```
python3 installer.py --reinstall <pathtoautosub> <pathtoconfigfile>
```

To use https you have to use a SSL key. The system expects the keyfiles (server.crt , server.csr , server.key) to be in the web2py directory. To generate keys and place them run the following (this can be skipped if you already have keyfiles you can use!):

```
./gen_keys.sh
```

To start the VELS_WEB daemon at port <port> and with web2py admin password <password> (this can be set by you!) run:

```
./daemon.sh start <port> <password>
```

After this step the VELS_WEB interface will be reachable via your browser at address:

```
https://<server_ip>:<port>/VELS_WEB
```

To stop the daemon just run the command:

```
./daemon.sh stop
```

8.7. General Configuration

Configuration items that can be changed dynamically are changeable in VELS_WEB -> General Config. These configurable items are:

- **Registration Deadline:** Users who are try to register after this deadline will get an error E-Mail.
- **Archive Directory:** Directory in which processed E-Mails are moved, this directory has to be present on the IMAP server!
- **Administration E-Mail:** E-Mail addresses which get question and system error E-Mails.

8.8. Whitelisting

Students that participate in the course have to be whitelisted in the VELS system. If the student tries to write an E-Mail to the system from an E-Mail address that is not on the Whitelist, an E-Mail with an error message is sent to him.

Whitelisting can be done in VELS_WEB under the tab *Whitelist*. E-Mail addresses can be added one at a time or multiple at a time (mass subscription). Removal of a single E-Mail address can also be done in the VELS_WEB. Names which will be used when users register can also be specified. This is usefull, because many users don't send E-Mails with their name configured in the "From" header.

8.9. Configuring the Tasks

Existing tasks can be assembled into a task queue. This configuration is done in VELS_WEB. Each task in the queue has to be created with the following properties:

- **TaskStart:** The start datetime for the task. The task will automatically be set to active if this datetime is reached. If auto_advance is active (configured via config file) users waiting for a task to become active will automatically receive an E-Mail with the task description for that task.
- **TaskDeadline:** The end datetime for the task. Submissions for a tasks after this datetime will be rejected.
- **TaskName:** The name of the folder with the implementation of the task in respect to the configured tasks_dir.
- **GeneratorExecutable:** The name of the generator executable.
- **Language**: Language code of the language in which the task discription for this task should be created.
- **TestExecutable:** The name of the tester executable.
- **BackendInterfaceFile:** Name of a backend interface file that offers a tester functions to control a backend. This allows easy switching between different simulator backends. Such scripts have to be stored in a directory `"_backend_interfaces"` in the configured tasks_dir.
- **Score:** The score a student gets for successful completion of the task. The scores for all completed tasks are added and can therefore also be used for grading.
- **TaskOperator:** The E-mail(s) of the operator of the task seperated by commas. These task operators are recipient of task specific questions ("Question Task N").
- **TaskActive:** The state of the task, for inactive tasks the generator won't be called. A Task is considered active when the current time is greater than the TaskStart

8.10. Notes on multiple VELS instances on the same machine

The following should be adhered if you want to use multiple VELS instances on one machine:

- Use different users for running the different instances. If you dont't do so you migth run into problems concerning the usage of the tmp directories of tasks in the test phase.
- Be sure to use different ports when starting the VELS_WEB daemon.

8.11. Configuration Checklist

1. Installed all needed libraries and tools for autosub, the tasks and VELS_WEB.
2. Configured the E-Mail server, including an E-Mail archive folder.
3. Created a config file for autosub.
4. Started the autosub system via `autosub.sh start <configfile>`.
5. Started VELS_WEB using the daemon.
6. Configured all parameters in General Config in VELS_WEB.
7. Configured all Tasks in VELS_WEB.
8. Whitelisted all students in VELS_WEB.

If you forget one of this steps or misconfigure, autosub tries to generate a meaningful message, still it's nicer to get everything running without being yelled at.

8.12. The Exam Mode

In Exam Mode the students additionally get sent a minimal testbench to test their design. To enable test mode, change the challenge-mode to exam in the config file and restart the autosub daemon.

Always check first if all the task you want to use support this mode!

The remaining configuration is similar to configuration for normal mode.

9. Creating a new task for the VELS system

The VELS system incorporates a tool to create a new task and a generalized method for testing the submitted code of a user. The goal of both is to keep the necessary effort for creating a new task to the minimum. This guide uses the VHDL Task Creator Wizard. If you do not want to use it or want to gain an insight on how the tasks system works in detail look at `autosub/doc/CreatingTasks.txt`.

9.1. VHDL Task Creator Wizard

The VHDL Task Creator Wizard is a graphical user interface to set up a new task. To use the wizard you will need python3-qt5 and python3-jinja2 installed on your system:

```
apt-get install python3-pyqt5 python3-jinja2
```

The tool itself can be found in the VELS system under the path `autosub/tasks/tools/vhdl_task_creator` and can be called by running `run.sh`. When started the wizard will guide you through a series of windows to set up your new task. In the first window you can enter:

- **Task name:** The name of the task. Used as the name of the task directory and in the task description for the user, the testbench_template, in the task.cfg file and in the comments of the scripts generateTestBench.py and generateTask.py.
- **Output directory:** The location where the task directory will be saved. In this directory a folder with the name of the task will be created.
- **User entities:** The name of the entities the user has to write the behavior for. The user will get the declaration of an entity in a file with the name [entity name].vhdl. And the user will write the behavior for the entity in a file with the name [entity name]_beh.vhdl
- **Extra files:** Additional VHDL packages or entities to be analyzed and elaborated in the order named here. For example the gates task uses these extra files to provide the user with gates defined in IEEE 1164.
- **Simulation Timeout (in s):** Time until the server hard stops the simulation of the user entities in seconds. Normally the simulation should be stopped from within the testbench well before the simulation timeout.
- **Description languages:** Choose to create skeleton LaTeX task description files for different languages. Skeleton templates are taken from the folder `"vhdl_task_creator/templates/task_description/"`.

- **Attach wavefile:** If the wavefile of the simulation shall be sent to the user.
- **Use task constraint script:** If a script shall be called to check the users entities before simulation. For example the pwm task checks here if the user did use the wait statement instead of using the clock signal.

The "next" button brings you to a window where you can configure the inputs and outputs of the users entity. If you have defined more than one user entity then they can be configured in a following window. When adding the inputs and outputs for the entity you can set the parameters:

- **Signal name:** The name of the signal. Any name conforming to the rules of the VHDL standard can be used here.
- **Signal type:** The available types are: std_logic_vector, std_logic, signed, unsigned and custom. Dependant on the selected type are the available configurations. For example the std_logic type leaves no configurations open except the signal name.
- **Length type:** The available options are: fixed, variable and single.
- **Length (Placeholder):** The bit-length of the selected signal type shall be entered here. If the length type is fixed then an integer greater 1 is expected. If the length type is variable then a string is expected as a placeholder. This placeholder needs to be replaced by your generateTask.py (e.g %%LENGTH or {{ length }}).
- **Resulting definition:** A preview of the resulting definition is displayed here as the parameters get configured.

When all inputs and outputs are configured a summary for the to-be-created task is shown in the last window. When the "finish" button is pressed the task will be created in the output directory.

The created structure of the task directory is:

- **templates:** Directory containing files which will be changed by the system for the task. These are the testbench_template.vhdl and the task_description_template.tex. The task_description_template is used for creating the description which is sent to the user. The testbench_template is used before starting a simulation to create a unique testbench, you will need to alter this file for your own task. If the length of an entity input or output was chosen to be variable, then this directory will also include the file [entity name]_template.vhdl.
- **static:** Directory containing files which will not be changed by the system itself. These are the files [entity name].vhdl and [entity name]_beh.vhdl.

The file [entity name].vhdl is the declaration of the entity for the user and [entity name]_beh.vhdl is where the user shall write the behavior of the corresponding entity. If the length of an entity input or output was chosen to be variable, then this directory will not include the file [entity name].vhdl.

- **scripts:** Directory containing python3-scripts which generate the unique task parameters for a user the task and fill the template files of the task (generateTask.py) and which generate the testbench for the created task parameters (generateTestBench.py). You will need to alter these files for your own task.
- **generator.sh** Script that is called when a task needs to be generated fo a user. Uses LaTeX to create the task_description PDF file for the user. Also specifies all needed files to the task description E-Mail for the user. You will need to alter this file, the comments in the file should guide you what you have to change.
- **task.cfg:** File containing bash variables for the generalized tester.sh to the specific task.
- **tester.sh:** Generalized bash script which sources the parameters of the task.cfg file and calls the functions of the backend interface. If you do not want to use the generalized tester you can write your own tester.
- **description.txt:** Contains the text which is sent to the user in the task description E-Mail.

9.2. Working with the generalized tester

The generalized tester provides a generalized method for testing the submitted code of a user. As mentioned one goal of this generalized testing method is to reduce the effort for creating a new task. Furthermore combining the testing for all tasks in one method allows to make future improvements at one centralized point.

When the VHDL Task Creator Wizard is used to create a new task (which is advised to do so) then, in the optimal situation, you will not have to directly deal with the generalized tester. When you are not using the Task Creator Wizard, or have to make further changes down the road, then the interface to work with the generalized tester is the task.cfg file. The task.cfg file is a configuration file for the tester which is located in each task directory. It gets sourced by the tester.sh script. The tester.sh script also sources the generalized tester and the task constraint check if applicable and calls functions defined in the backend interface file. The task.cfg file includes the configurations:

- **task_name:** The variable task_name is most importantly used in the generalized tester during elaboration and simulation of the submitted VHDL code. This is because the entity of the testbench is expected to be named [task_name]_tb.

- **userfiles:** The variable userfiles contains all the VHDL files expected to be submitted by the user. They are used to check if all files were submitted by the user and further on when the user VHDL files are analyzed.
- **entityfiles:** Variable containing the filenames of the entity files. The entity files contain the declaration of the user entities. These entity files are not from the user. The user writes the behavior of the entities in the user files.
- **extrafiles:** Variable containing the filenames of the extra entity files. Those extra VHDL entities are not from the user, but are supplied to the user in the task description. The extra files are sent to the user to provide ready coded VHDL entities the user can use in his task. Extra files are expected to be in the `"static/"` folder of the task.
- **constraintfile:** The constraintfile variable contains the path to the constraint file in the task directory. The constraint file is sourced in the tester.sh script which is located in the task directory.
- **simulation_timeout:** The simulation timeout is used during the simulation of the users VHDL code. If the simulation timeout is hit then the simulation process is killed.
- **attach_wave_file:** If set to '1' then the wavefile of the simulation will be attached in case the simulation of the users code was not successful.

10. Appendix

10.1. Config file fields

The configuration file can include the following groups and fields. Bold fields are mandatory, all other fields can be ommited. Default paths are in respect to the `autosub/src` directory, if you set paths explicitely they have to be absolute.

[imapserver]

Field	Description	Default
servername	Hostname.domain of the IMAP server.	
username	Username to be used to login.	
password	Password to be used to login.	
email	E-Mail address to be used at the IMAP server.	
security	Security protocol to be used when connecting. Possible values: none ssl starttls	ssl
serverport	Port to be used.	ssl:993 else:143

[smtpserver]

Field	Description	Default
servername	Hostname.domain of the SMTP server.	
username	Username to be used to login.	
password	Password to be used to login.	
email	E-Mail address to be used at the SMTP server.	
security	Security protocol to be used when connecting. Possible values: none ssl starttls	starttls
serverport	Port to be used.	ssl:465 starttls:587 none:25

[system]

Field	Description	Default
num_workers	Number of Worker threads. This influences how many tests can conducted in parallel.	
queue_size	Size of the thread communication queues.	
poll_period	Period in seconds at which the mailbox is checked for new E-Mails from students.	60
semesterdb	The name and path of the semester database.	semester.db
coursedb	The name and path of the course database.	course.db
log_dir	Directory to put tasks and autosub log files.	logs

log_threshhold	Threshhold from which level up log entries should be logged. Possible values: DEBUG INFO WARNING ERROR	INFO

[**course**]

Field	**Description**	**Default**
specialmsgs_dir	Directory for messages that are sent to users on special events.	SpecialMessages
tasks_dir	Directory of the task implementations.	
course_name	The name of the course.	No name set
mode	The mode in which to run. Possible values: exam normal	normal
allow_requests	Decides if the system runs in request queue mode or linear queue mode. If this is activated auto_advance is set to no. Possible values: once multiple no	no
auto_advance	Decides if users get auto advanced to a task which is activated by its TaskStart. Only possible in linear task queue mode. Possible values: yes no	no

10.2. Description of the message queues

job_queue The job_queue is used to trigger Worker threads to start testing solutions that were received. The messages in the job_queue are comprised of the following fields:

- **user_id:** The unique User ID of the user who submitted this solution.
- **user_email:** The E-Mail address of the user who submitted this solution.
- **task_nr:** Number of the task that solution has been submitted for.
- **message_id:** The unique message-id of the user E-Mail on the IMAP server, which started the testing.

sender_queue If a thread wants to send an E-Mail to a user, the Sender thread is notified via this queue. The data needed by the Sender thread is in the messages in the sender_queue. These messages consist of the following fields:

- **user_id:** The unique User ID of the user who shall receive this E-Mail.
- **recipient** E-Mail address of the recipient (the content of the 'To' field in the E-Mail).

- **message_type** The message type is used to decide on how to format the E-Mail, and whether or not additional E-Mails have to be sent out (e.g. when a question is handled). Static messages that are sent to users implied by the message type can be found in the directory SpecialMessages. Possible values for message_type are:
  - `Task` – A task description.
  - `Success` – A submission for a task has been tested and returned success.
  - `Failed` – An error message for a failed test-run of a submission.
  - `SecAlert` – Scanning the code revealed that this might be an attack on the system, a message to the administrators.
  - `TaskAlert` – An error message to administrators for failures in task specific files.
  - `TaskErrorNotice` – An error message to users if a generator or tester has an error in respone to a user message.
  - `InvalidTask` – A submission or request for a non-existent task.
  - `Usage` – Message with system usage explanation for the user.
  - `Question` – Confirm that a question was received.
  - `QFwd` – Forward a question to the administrator.
  - `Welcome` – Welcome message to a new user.
  - `NotAllowed` – A user who is not on the whitelist sent a mail to the system.
  - `TaskNotSubmittable` – A user wants to submit to a task for which he has not received a task description yet.
  - `TaskNotActive` – Response to an user action concerning a task that is not active yet.
  - `Status` – Message containing the current status of a user in the course.
  - `TaskList` – List of all tasks configured in the course.
  - `RegOver` – Whitelisted user tries to register after registration end datetime.
  - `NoMultipleRequest` – User requests task he already received and course is not configured for multiple request.
  - `CurLast` – Message indicating all active tasks have been solved sucessfully and the next task has not started yet.

- `DeadTask` – User submits to task wit deadline passed.

- **body** Body of the E-Mail that triggered the message to be sent to user/moderator/admin.
- **task_nr** Number of the task that message concerns (e.g. the current one if the test failed, the next one if the task description shall be sent).
- **message_id:** The unique message-id of the user E-Mail on the IMAP server, which triggered the message sending.

logger_queue Trigger the logger to write a log message about an event that happened.

- **message:** The text that describes the event that shall be logged.
- **level:** The log-level of this log message; available log-levels are DEBUG, INFO, WARNING and ERROR.
- **src:** Name of the thread that reported the event that shall be logged.
- **dst:** Log destination (autosub log, task message, task error).

generator_queue The generator_queue is used to trigger the Generator thread to generate a new (variant of) a task. That means that certain parameters of the task are randomized, in order to assure that each student receives his/her very own example.

- **user_id:** The unique UserID of the User for whom the task is generated.
- **user_email:** E-Mail address of the User for whom the task is generated.
- **task_nr:** Number of the task that shall be generated.
- **message_id:** The unique message-id of the user E-Mail on the IMAP server, which triggered the task generation.

archive_queue The archive_queue is used to announce that a E-Mail has been finished processing. This triggers archiving and removing task submissions from the active queue. own example.

- **message_id:** The unique message-id of the E-Mail on the IMAP server which shall be archived.
- **is_finished_job:** Flag to show that the message-id belongs to a finished processed submitted task.

10.3. Semester database semester.db

The database semester.db contains the following tables (and entries in those tables):

Table Name	Users
Description	Used to collect all necessary information on the Students.
Table Entries	• **UserId**: A unique UserID given at registration time (when the first E-Mail is received from the users E-Mail address). • **Name**: The name of the user as specified in the "from" field of the E-Mail. • **Email**: The E-Mail address of the user as specified in the "from" field of the E-Mail. • **RegisteredAt**: Timestamp of the registration of the user. • **LastDone**: Timestamp of the E-Mail that contained the successful solution of the last task (only if this user has already finished the last task). • **CurrentTask**: The task the user is currently working on.

Table Name	TaskStats
Description	Contains one entry for each available task, collecting statistics on the individual tasks.
Table Entries	• **TaskId**: Unique ID of the task. • **NrSubmissions**: Number of solutions received for the task with this TaskId. • **NrSuccessful**: Number of correct solutions received for the task with this TaskId.

Table Name	TaskCounters
Description	Implements counters for certain events, examples for such events are: E-Mail received, E-Mail sent, question received, new user.
Table Entries	• **CounterId**: Unique ID of the counter. • **Name**: Name of the counter. • **Value**: Value of the counter.

Table Name	UserTasks
Description	Used to map configured tasks to users – e.g. store the generated examples so they can be fetched later on for testing.
Table Entries	• **TaskNr**: Unique number of the task. • **UserId**: Unique ID of the user – the combination of TaskNr and UserId make the entry unique. • **TaskParameters**: Either the parameters that describe the setting for this particular student, or a value that can be used to derive the parameters from. • **TaskDescription**: Message that describes the task that was specifically generated for the student. and shall be sent to the student. • **TaskAttachments**: List of attachments that shall be sent to the student (path+filename). • **NrSubmissions**: Number of submissions the student has done for this task. • **FirstSuccessful**: Number of the first successful submission.

Table Name	SuccessfulTasks
Description	Used to save the task numbers which a user has successfully completed.
Table Entries	• **UserId**: ID of the user – the combination of TaskNr and UserId make the entry unique. • **TaskNr**: Number of the task. – the combination of TaskNr and UserId make the entry unique.

Table Name	UserWhiteList
Description	A Whitelist of E-Mail addresses that shall be authorized to interact with the system.
Table Entries	• **UniqueId**: Unique ID in the whitelist table. • **Email**: E-mail address that shall be authorized to interact with the system. • **Name**: The name the user shall be assigned instead of the value from the "From" field of the registration E-Mail.

10.4. Course database course.db

The database course.db contains the following tables (and entries in those tables):

Table Name	TaskConfiguration
Description	Used to configure tasks, including their order, the scripts used to test submissions, etc.
Table Entries	• **TaskNr**: The unique number of the task – this number is used to establish the order of tasks as received by the students. • **TaskStart**: Timestamp of when the task shall be available for students (if any). • **TaskDeadline**: The deadline until which the task has to be successfully submitted (if any). • **TaskName**: The name of the task folder of te task in respect to the configured tasks_directory. • **GeneratorExecutable**: The executable (script, binary, etc.) used to generate unique examples for each student. If this is not set, all students will receive the same task. • **Language**: Language code of the language in which the task discription for this task should be created. • **TestExecutable**: The executable (script, binary, etc.) used to test the results submitted by the students. • **BackendInterfaceFile**: Name of a backend interface file that offers a tester functions to control a backend. • **Score:** The points the student scores by solving this task. • **TaskOperator:** The E-Mail address of the course operator, who is responsible for this task.

Table Name	SpecialMessages
Description	A collection of texts that shall be sent, in the case special events.
Table Entries	• **EventName**: Name of the event the SpecialMessage is related to. Naming is uppercase of the related message_type in the sender_queue. • **Text**: The text that shall be sent in case of the event **EventName** occurs.

Table Name	GeneralConfig
Description	Store some general configuration to be accessible for the VELS_WEB interface.
Table Entries	• **ConfigItem**: Name of the Configuration Item. • **Content**: Content of the Configuration Item.

10.5. Handling of concurrent testing

Handling of testing multiple submissions from users is handled in autosub in the following ways:

- Multiple Worker threads and seperated user tasks directories make it possible that multiple task submissions can be processed at the same time.
- Active task submissions are put in an active queue and removed once they have been fully processed and an response has been sent to the student. Active task submissions that are not processed after 5 minutes are presumed dead and removed from the active queue.
- To prevent conflicts only one combination of user and task_nr is allowed to be active at any time. If the user submits a solution for the same task before it has been fully processed, it will be put in a backlog queue. Tasks from backlog are made active once the conflicting submission has been fully processed.

Abbreviations

IMAP Internet Message Access Protocol. 7

SMTP Simple Mail Transfer Protocol. 7, 8

VHDL VHSIC Hardware Description Language. 2

VHSIC Very High Speed Integrated Circuit. 2, 40

References

[Coo03] Jim Cooling. *Software Engineering for Real-Time Systems*. Addison Wesley, first edition edition, 2003.

[DeM81] Tom DeMarco. *Structured Analysis and System Specification*. Yourdon P.,U.S., april 1981 edition, 1981.

[Goo01] Hassan Gooma. *Software Design Methods for Concurrent and Real-Time Systems*. Addison Wesley, fifth printing edition, 2001.