Email to 3D Print

Chad Klusek
Andrew Ward
Evan Zeller

CS499 - Senior Design Project
University of Kentucky
Fall 2014

Sections

Section 1: Disclaimer

Disclaimer:

This project has been designed and implemented as a part of the requirements for CS-499 Senior Design Project for Fall 2014 semester. While the authors make every effort to deliver a high quality product, we do not guarantee that our products are free from defects. Our software is provided "as is," and you use the software at your own risk.

We make no warranties as to performance, merchantability, fitness for a particular purpose, or any other warranties whether expressed or implied.

No oral or written communication from or information provided by the authors or the University of Kentucky shall create a warranty.

Under no circumstances shall the authors or the University of Kentucky be liable for direct, indirect, special, incidental, or consequential damages resulting from the use, misuse, or inability to use this software, even if the authors or the University of Kentucky have been advised of the possibility of such damages.

Section 2: Abstract

3D printing is now hitting its stride as a consumer level product with printers like the MakerBot Replicator Mini[1] and the Replicator2[2] now being affordable and easy to use. Hackers and geeks around the world have experimented and improved upon the hardware and software that drive this industry. However, there still are not many tools, especially for the automation of printing. We have attempted to implement an automated 3d printing build process to print any file to your printer from anywhere in the world.

Section 3: Introduction

The project commissioned by our client, the Collexion Hackerspace[3], is to provide a program (written in Python), that acts as a pipeline for converting a valid 3D model, received via an email interface, into G-code instructions that can be interpreted by a MakerBot Replicator 2 3D printer. They use 3D printing as an educational tool for The Learning Center[4], a Fayette County Schools project in Lexington, Kentucky. They have felt that their current build process is inefficient, requiring manual conversion between and through multiple file formats and physically carrying data from the computer to the printer. This project aims to limit human interaction in the 3D printing process, by providing powerful tools to do the compilation and printing work, and an easy to use email interface that keeps the user informed about their print job's status. The only human interaction beyond an email submission, will be the removing of the previously printed object from the build platform, and verification that the system is ready for the next build by a local administrator.

Section 4: Product Specifications

The following bulleted list provides a summary of project specifications and interesting features to be completed, in order of highest to lowest priority, and as time allows. In summary, the program should:

- Receive a user submitted email, and verify that it contains an OBJ or STL formatted attachment.
- Convert an OBJ model to STL format
- Validate that the model can be printed
- Slice the model (convert to G-code)
- Print or queue the model, only when the printer status is OK
- Monitor or poll for printer status
- Log Print job status
- Handle as many 3d model types as possible, especially proprietary formats.
- Send a notification email to a user if there is a problem with the printer or their job
- Detect if the printer is in an OK state automatically so the next job can be printed

Section 5: Effort and Size Estimation

During the project design phase, the intermediate COCOMO method[5] was used to estimate the number of PERSON-MONTHS required for this project. The number of PERSON-MONTHS, E, is computed using the formula:

$$E = a_i(KLOC)^{(b_i)} * EAF$$

Assumptions and Estimations:

This is an "Organic Project", with little or no pre-existing code base, and a small team of developers working on the project. Therefore, a_i = 3.2, b_i = 1.05

KLOC is "kilo lines of code". This is a very rough, uneducated guess, but the "gut feeling" is that this project is closer to 1000 lines than 2000 or more. Therefore, KLOC = 1.

Next, we calculate the EAF (effort adjustment factor) by rating a number of categories from very low to very high, and providing the appropriate adjustment from a table. The components of the EAF calculation are:

Required software reliability: 1.15
Size of application database: 0.94
Complexity of product: 1.00
Run time constraints: 1.00
Memory constraints: 1.00
Volatility of VM environment: 1.00
Required turnabout time: 1.15
Analyst capability: 1.19
Applications experience: 1.13
Software engineer capability: 1.17
Virtual machine experience: 1.00
Programming language experience: 1.00
Application of SW engineering methods: 1.10
Use of software tools: 1.10
Required development schedule 1.10

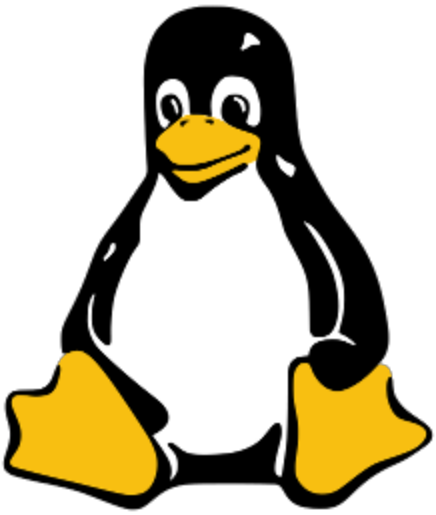
EAF is calculated as the product of the above 15 estimates, which is ~2.60
So:
#PERSON-MONTHS = $3.2(1)^{1.05}*2.60$ = **~ 8.3 PERSON-MONTHS**

Unexpected occurrences, like the lack of software that performs the tasks that we expected initially can put significant restraints on the project. For instance we attempted to use at least five different tools in the printing stage of the pipeline. Each of these tools presented their own complications from incompatibility with makerware's proprietary protocols to ungraceful failure and lack of documentation.

Section 6: Schedule and Milestones

15 September	Project assignment and team formation
17 September	Initial meeting with customer
17 September	Start of module design
24 September	Start of module code
03 October	Project website started
20 October	Completion of module design
24 October	Midterm presentation
03 November	Start of test plan development
06 November	Testing session at CirrusMio
10 November	Finalized test plan
18 November	Testing session at CirrusMio
20 November	Testing session at CirrusMio
03 December	System testing at Learning Center, customer delivery
12 December	Final presentation
15 December	Delivery of report to Todd Willey
17 December	Delivery meeting with Dr. Piwowarski

Section 7: Platforms, Tools, and Languages

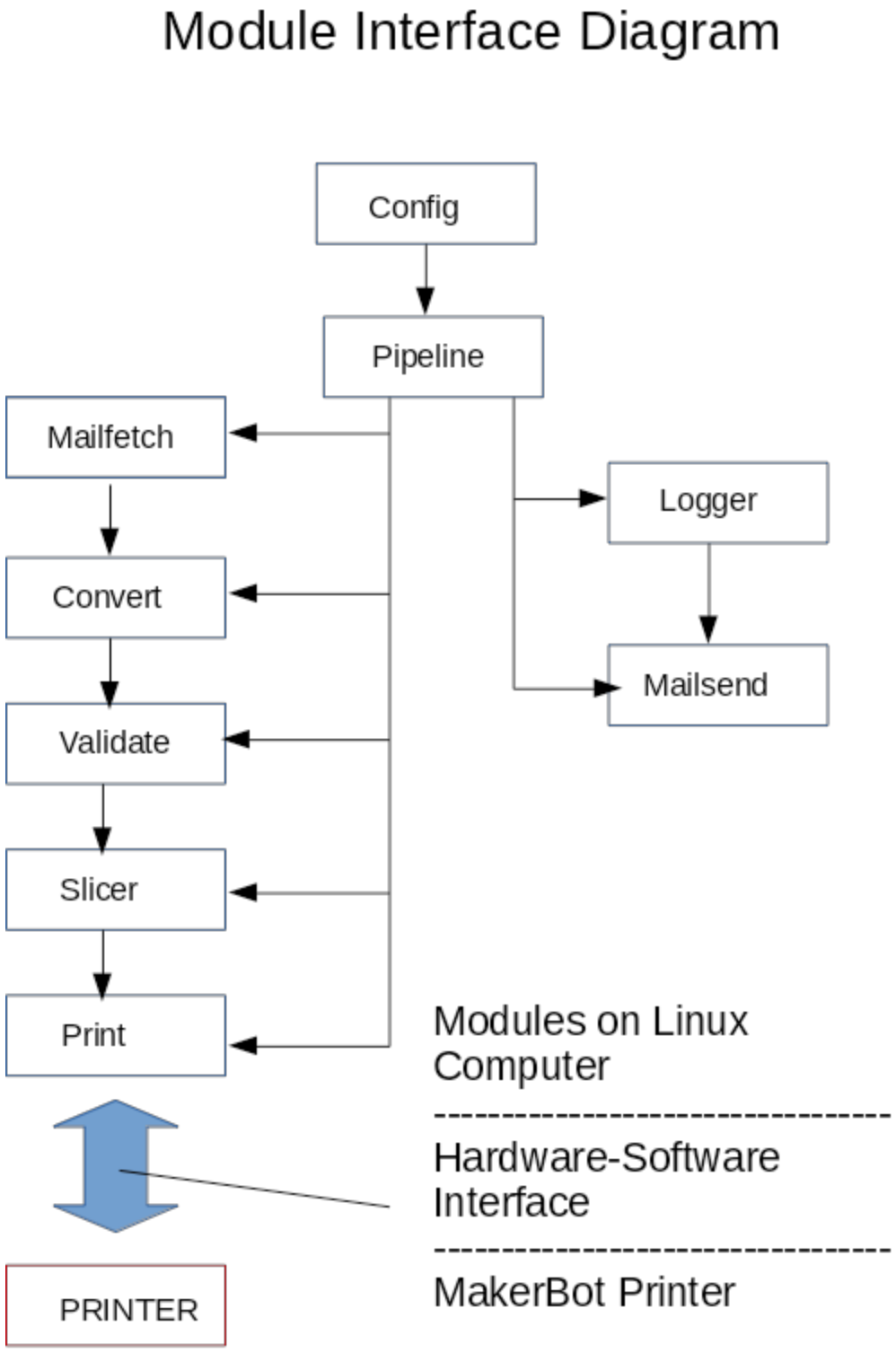


The application was developed to reside on a workstation, computer, or server to be provided by the customer. The operating system is a non-specific Linux distribution or virtual machine. This platform was chosen for us, but makes sense. We don't need any of the user facing features that a Windows or Mac platform would provide, since most of the time the computer will only be used by an admin who will understand the system.

We used python 3 as the implementation language, to glue various 3rd party code together. We selected this language because it is high level and easy to read and debug, and is universally available. We didn't need much performance, since the printer itself would always be our bottleneck, so we weren't willing to sacrifice expressive power for a "larger" language like c or java.

Various open source projects already exist to perform slicing and to act as a "printer driver". We used several of them, including Slic3r and GPX. Using open source code was important to us, because there are some complicated bureaucratic rules on using commercial products for a project by a public school.

Section 8: Design



Section 8.1: Module Descriptions

Pipeline()

Purpose:

Provide the infrastructure of a PrintJob object to encapsulate data pertinent to each given print request, such as sender, status, file paths, ...

Inputs:

No explicit inputs

Outputs:

A PrintJob object

Design:

In the spirit of object oriented methods, this module is a class with constructor, getters, setters, etc...

Tracks the current pipeline stage (block diagram phase completed)

Contains sender information

Contains system related info such as timing and file paths

Config()

Purpose:

Read/write configuration values from/to a configuration file on disk

Inputs:

Read or write flag indicating action to be performed

Outputs:

Read configuration from disk or writes configuration to disk

Design:

If flag is READ, read configuration from disk

IF flag is WRITE, write configuration data to file on disk

Mailfetch()

Purpose:

Fetch STL or OBJ attachments sent via email by users

Inputs:

Module configuration parameters

Outputs:

Save attachments to disk

Create a PrintJob object containing attachment and sender email address

Design:

Read configuration data from a file on disk

Open a socket to the Google mail server

Login to the e-mail account

Generate a list of unread messages from the inbox

For each message containing a valid attachment:
- Capture the e-mail address of the sender
- Save the attachment to disk
- Create a PrintJob object to pass to later modules

Logout of e-mail account and close connection socket
After a number of minutes have elapsed, return to step 2

Convert()
Purpose:
Convert OBJ to STL format model
Inputs:
OBJ file
Outputs:
STL file
Design:
If file extension is .stl, do nothing
else if file extension is .obj convert to .stl
else error condition

Validate()
Purpose:
Verify that STL model is valid/printable
Inputs:
STL file
Outputs:
Error if file is not valid or printable
Design:
Execute a sequence of validation routines
If any return a failure condition log the error and send email to user

---

Slicer()
Purpose:
Produce G-code, which provides low level instructions allowing the target hardware to print the intended object, layer by layer.
Inputs:
An STL format model file
Outputs:
G-code if slicing is successful
Error log and email, otherwise
Design:

Call slic3r or replicatorG with appropriate arguments for slicing
If error:
Log and send email to user
Else:
Save G-code in file to be sent to hardware for printing

Print()

Purpose:
Physically print the requested model

Inputs:
File containing G-codes

Outputs:
Hardware prints the requested design or report error is failure occurs

Design:
Call replicatorG to print the model
If error:
Log and send email to user
Else:
Model was printed successfully
Notify user by email that job was successfully completed
Log job completion and update quota statistics

Logger()

Purpose:
Write a message to the error log

Inputs:
Pipeline stage
User/sender email address
Error message

Outputs:
Write time, user name, stage, and error message to an error log file

Design:
Open log file for writing
Write time, user name, stage, and error to log file
Close log file

Mailsend()

Purpose:

Construct and send an email to the user advising of success or failure

Inputs:

Pipeline stage

Output:

Email to the user informing them of success, or failure and the point of failure

Design:

Construct a properly formed header
Construct a “form letter” body based on value of the pipeline stage provided
Send the message to user

Section 8.2: User Scenarios (Use Cases)

Case 1 - Normal Case
Here is an example of an anticipated normal user scenario:
1. User submits a .stl or .obj model to be printed to a Gmail account designated for such purpose.
2. Program detects than an unread message has arrived in the INBOX.
3. The attachment is a recognized model file format, so the file is saved to disk, and the user name of the sender is captured.
4. Program verifies that the model file is valid and printable.
5. Model file is converted to G-code.
6. G-code is sent via usb interface and replicatorG software to the Makerbot printer.
7. Model is printed. Job statistics are logged, and user is notified by email that the print job completed successfully.

Case 2 – Print job failed
Here is an example of a failure to print by the hardware:
1. User submits a .stl or .obj model to be printed to a Gmail account designated for such purpose.
2. Program detects than an unread message has arrived in the INBOX.
3. The attachment is a recognized model file format, so the file is saved to disk, and the user name of the sender is captured.
4. Program verifies that the model file is valid and printable.
5. Model file is converted to G-code.
6. G-code is sent via usb interface and replicatorG software to the Makerbot printer.
7. Model did not print.
8. Error is logged and user is notified of the failure and reason for failure.

Case 3 – Model file is invalid
Here is an example of a scenario where the submitted model is invalid or not printable:
1. User submits a .stl or .obj model to be printed to a Gmail account designated for such purpose.
2. Program detects than an unread message has arrived in the INBOX.
3. The attachment is a recognized model file format, so the file is saved to disk, and the user name of the sender is captured.
4. Program examines that the model file is invalid.
5. Error is logged and user is notified of the failure and reason for failure.

Case 4 – Model file is not STL or OBJ

In this example, the program did not receive an STL or OBJ file, due to a mistake or misunderstanding by the target user(s), a spam email that bypassed the Gmail filters, or due to some other unanticipated reason:
1. User submits an unsupported model to be printed to a Gmail account designated for such purpose.
2. Program detects than an unread message has arrived in the INBOX.
3. The attachment is not a supported model file format, so no further action is taken.

Section 9: Implementation

We chose to implement the pipeline as described by the customer using Python 3 in a Linux environment. As 3D printing has historically been closely tied with the open source community there are a variety of tools available to perform the actions required by this project. There were some unexpected consequences of relying on these open source tools to perform the actions we needed that we will discuss in further detail.

The python libraries used to fetch mail from inboxes were found to be robust and performed as expected. Moving further into the pipeline we begin making use of more blackbox tools via bash. The focus here is on inputs and outputs. While the initial project description only called for the ability to slice[6] STL files we, as well as the customer, decided to include a step to allow other 3D modeling formats to be supported. Although STL and OBJ model formats are supported, all submitted models are converted to STL format to minimize test cases and possible support issues.

For conversion of OBJ models to STL format, we explored both meshconv and meshlab. Meshconv has a precompiled linux binary executable available, and most testing was performed with meshconv, For that reason it was used for conversion purposes during the final testing session at the Learning Center, and during the final presentation. If in the future it is desired to support more model formats, meshlab offers more import and export options than meshconv.

Slic3r is the most robust tool for slicing 3D model files into G-code[7]. G-code operates as machine instructions for the 3D printer, using 3D coordinates to extrude plastic layer by layer constructing the target object. Transferring these instructions to the printer is where our struggle begins.

There are many open source tools for communicating G-code to a 3D printer. Makerbot printers now run firmware that communicates using a proprietary protocol. This makes the field a little bit thinner. What follows is a list of the different tools and approaches we tried and researched for this last piece of the pipeline:

1. ReplicatorG: ReplicatorG was the first option we researched, while it is robust and is the open source standard in 3D printing it lacks a programmatic interface and thus cannot be used reliably.
2. DuplicatorG: A fork of ReplicatorG that gives ReplicatorG a programmatic CLI interface. While promising the project was out of date and didn't seem to be maintained.
3. Printrun/Pronsole.py: While also promising this option requires a custom firmware to be installed on the printer, which is not an option for our customer.
4. Makerware/Conveyor: Makerware software uses a service known as conveyor to communicate with the printer. It may be possible to develop a shim to communicate with the conveyor service, but there are no documented interfaces and this option would take more time than we have available to reverse engineer and implement.
5. Roll your own driver: similar to the conveyor shim this is not an option in the timespan we have, in itself it could be a semester long project to implement a driver for the x3g protocol.

6. Octoprint: One of the remaining viable options octoprint is a modern piece of software that runs in the web browser and communicates with printers on the network or connected directly. May require custom firmware, which is not an option for us. Will require a modified version of pyserial on the system.
7. GPX: The most promising tool is gpx, which is what the pipeline currently uses for communicating with the printer. While the print job is seen on the printer's display the print job makes no progress towards completion due to an unknown and undocumented bug.

We believe that at this point octoprint presents the best possible solution for the final part of the pipeline. Its modern interfaces integrate well with software best practices and trends. While the time remaining may not prove to be enough for us to implement an interface to octoprint we would recommend it as the first in line solution for the printing part of the pipeline.

Section 9.1: Unit Testing

Unit tests include test cases specific to the major modules of the project, as well as functions contained within each individual module. Our plan for unit testing our project will involve (ideally automated) testing of the individual parts of the project. If they each pass the test cases, we can be confident that they behave as expected.

Config.py

Number	Description	Input	Expected Output
2.1.1	Normal Case	Reads configuration file	Configuration variable values available for other modules to use as needed
2.1.2	Error Case	Configuration file malformed	Exit with informative error message
2.1.3	Error Case	Configuration file not found	Exit with informative error message

Converter.py

Number	Description	Input	Expected Output
2.2.1	Normal Case	STL model	No conversion performed
2.2.2	Normal Case	OBJ model	Convert model to STL format
2.2.3	Error Case	Any other file type	No conversion, log error
2.2.4	Error Case	File has STL or OBJ extension, but is another file type	No conversion, log error
2.2.5	Error Case	OBJ file conversion failed	Log error

Logger.py

Number	Description	Input	Expected Output
2.3.1	Normal Case	Log file found	Log file available for update
2.3.2	Error Case	Log file not found	Raise exception and exit

Mailfetch.py

Number	Description	Input	Expected Output
2.4.1	Normal Case	Reads configuration file	Configuration variable available to Mailfetch
2.4.2	Normal Case	No unread messages found	No further actions, no PrintJob
2.4.3	Normal Case	Unread message, no attachment	No further actions, no PrintJob
2.4.4	Normal Case	Unread message with 1 STL or OBJ attachment	Create PrintJob object Save attachment to disk Save sender username
2.4.5	Normal Case	Unread message with >1 STL or OBJ attachment	Create PrintJob for each attachment Save attachments to disk Save sender username
2.4.6	Error Case	Configuration variable missing	Raise exception and exit Email admin and log error
2.4.7	Error Case	Opening connection fails	Raise exception and exit Email admin and log error
2.4.8	Error Case	Login to gmail fails	Raise exception and exit Email admin and log error
2.4.9	Error Case	Unable to write attachment to disk	Raise exception and exit Email admin and log error

Mailsend.py

Number	Description	Input	Expected Output
2.5.1	Normal Case	Configuration file read	Variables available to Mailsend
2.5.2	Normal Case	Receiver address Pipeline Stage	Send email to receiver with contents depending on stage value
2.5.3	Error Case	Configuration variable missing	Exit with informative error
2.5.4	Error Case	Opening connection fails	Raise exception and exit Email admin and log error
2.5.5	Error Case	Login to gmail fails	Raise exception and exit Email admin and log error
2.5.6	Error Case	Message send fails	Raise exception and exit Email admin and log error

Pipeline.py

Number	Description	Input	Expected Output
2.6.1	Normal Case	Instantiated PrintJob	Stage updated based on pipeline point Contains member variables: Sender Filenames Time Status

Printer.py

Number	Description	Input	Expected Output
2.7.1	Normal Case	Reads configuration file	Config variables available for other modules
2.7.2	Normal Case	PrintJob with successful printing	Admin removes object from build plate Admin places object in box for customer Successful job logged Email sent to user
2.7.3	Error Case	PrintJob not printed	Log error Email admin and user
2.7.4	Error Case	Configuration variable missing	Exit with informative error
2.7.5	Error Case	File to be printed not found	Log error Email admin and user

Slicer.py

Number	Description	Input	Expected Output
2.8.1	Normal Case	Reads configuration file	Config variables available for other modules
2.8.2	Normal Case	STL model, correctly sliced	G-code file
2.8.3	Error Case	STL model, slicing error	Log error Email user
2.8.4	Error Case	Configuration variable missing	Raise exception and exit Email admin and log error

Note: Due to using slic3r for validation as well as slicing, section 2.9 validator.py tests presented in the original testing plan have been removed.

Section 9.2: Integration Testing

Integration tests examine the behavior between interacting modules of the project. The Main.py module functions much like a driver, calling other program modules when they are needed. This kind of testing will simply make sure that the data flows between modules according to the spec.

Number	Description	Input	Expected Output
3.1.1	Normal Case	Completed print job	A part is successfully printed Email sent to job originator with news of success
3.1.2	Normal Case	Failure at print stage	No part was printed Email job originator of print stage failure Clean up temp files
3.1.3	Normal Case	Failure at slicing	No part was printed Email job originator of slicing failure Clean up temp files
3.1.4	Normal Case	Failure at validation	No part was printed Email user of validation failure Clean up temp files
3.1.5	Normal Case	Failure at conversion	No part was printed Email job originator of conversion failure Clean up temp files
3.1.6	Error Case	Config module not present	Raise exception and exit Email admin and log error
3.1.7	Error Case	Logger module not present	Raise exception and exit Email admin and log error
3.1.8	Error Case	Mailfetch module not present	Raise exception and exit Email admin and log error
3.1.9	Error Case	Converter module not present	Raise exception and exit Email admin and log error
3.1.10	Error Case	Validator module not present	Raise exception and exit Email admin and log error
3.1.11	Error Case	Slicer module not present	Raise exception and exit Email admin and log error
3.1.12	Error Case	Print module not present	Raise exception and exit Email admin and log error
3.1.13	Error Case	Mailsend module not present	Raise exception and exit Email admin and log error

Section 9.3: System and Customer Testing

This is the final phase of testing, and its purpose is to verify functionality at the customer site, on customer provided hardware, using a customer installed Linux distribution and Python 3.x environment. The integration test cases will be repeated, in the customer's production environment.

Number	Description	Input	Expected Output
4.1.1	Normal Case	Completed print job	A part is successfully printed Email sent to job originator with news of success
4.1.2	Normal Case	Failure at print stage	No part was printed Email job originator of print stage failure Clean up temp files
4.1.3	Normal Case	Failure at slicing	No part was printed Email job originator of slicing failure Clean up temp files
4.1.4	Normal Case	Failure at validation	No part was printed Email user of validation failure Clean up temp files
4.1.5	Normal Case	Failure at conversion	No part was printed Email job originator of conversion failure Clean up temp files

Section 9.4: Final Test Results

Number	Date	Pass/Fail	Notes
2.1.1	11/29/14	P	
2.1.2	11/29/14	P	
2.1.3	11/29/14	P	
2.2.1	11/29/14	P	
2.2.2	11/29/14	P	
2.2.3	11/29/14	P	
2.2.4	11/29/14	P	
2.2.5	11/29/14	P	
2.3.1	11/29/14	P	
2.3.2	11/29/14	P	
2.4.1	11/29/14	P	
2.4.2	11/29/14	P	
2.4.3	11/29/14	P	
2.4.4	11/29/14	P	
2.4.5	11/29/14	P	
2.4.6	11/29/14	P	
2.4.7	11/29/14	P	
2.4.8	11/29/14	P	
2.4.9	11/29/14	P	
2.5.1	11/29/14	P	
2.5.2	11/29/14	P	
2.5.3	11/29/14	P	
2.5.4	11/29/14	p	
2.5.5	11/29/14	P	

Number	Date	Pass/Fail	Notes
2.5.6	11/29/14	P	
2.6.1	11/29/14	P	
2.7.1	11/29/14	F	Stub code for print module
2.7.2	11/29/14	F	Not Automated, stub code
2.7.3	11/29/14	F	Stub code for print module
2.7.4	11/29/14	F	Stub code for print module
2.7.5	11/29/14	F	Stub code for print module
2.8.1	11/29/14	P	
2.8.2	11/29/14	P	
2.8.3	11/29/14	P	
2.8.4	11/29/14	P	
Number	Date	Pass/Fail	Notes
3.1.1	12/1/14	F	Performed with stub since automated print not implemented
3.1.2	12/1/14	P	
3.1.3	12/1/14	P	
3.1.4	12/1/14	P	
3.1.5	12/1/14	P	
3.1.6	12/1/14	P	
3.1.7	12/1/14	P	
3.1.8	12/1/14	P	
3.1.9	12/1/14	P	
3.1.10	12/1/14	P	
3.1.11	12/1/14	P	
3.1.12	12/1/14	P	
3.1.13	12/1/14	P	

Number	Date	Pass/Fail	Notes
4.1.1	12/3/14	F	Unable to perform automated print with GPX
4.1.2	12/3/14	F	Stub code since printer.py incomplete
4.1.3	12/3/14	P	
4.1.4	12/3/14	P	
4.1.5	12/3/14	P	

Section 10: Future Enhancements

There are many opportunities to improve and enhance the functions of the Email to 3D Print program. The first opportunity is to introduce use statistics. This feature is not currently present, but would be valuable in future versions, as it could be used by the Learning Center in grant proposals, and justification for financing.

Another possibility is to introduce quota enforcement, where a user would only be pre-authorized to print a specified number of jobs.

Enhancements to security would also provide a great opportunity for future work. In this first project iteration, the customer has expressed interest in the proof of concept, and not in security, and the development team has not introduced security features by design. As a result, anyone in the world who sends email to the designated account would invoke a print job of a valid model, or, more maliciously, could introduce a possible denial of service by frequently sending any email to the account. A user whitelist could potentially alleviate some of the security issues, and a lockout or shutdown feature would probably be helpful as well.

In its current form, error handling is performed by both a logger module, and an email to the job originator that states either success, or the stage of failure (fetch, conversion, validation, slicing, or printing). This behavior alone is desirable and useful, but the error handling would improve with more specific information.

Individualized configuration provides another potentially desirable future project enhancement. Currently, low level parameters such as plastic wire diameter, target heating temperatures, and flow rates, among others, are set to sane defaults, but in some specialized cases, it may be of interest to the user to customize those based on individual model designs, or lessons learned through previous print jobs (or failures). This could potentially be accomplished by allowing the user to also submit a parseable configuration file with their print job.

A final consideration is the management of dependencies for helper programs, such as Slic3r, and the numerous print driver programs that were examined. This is currently a server side, hands on process that may involve pulling code from a project's git repository, manual dependency checking, and manual compilation in some cases.

Section 11: Maintenance

During the final testing visit to the Learning Center, we learned of a recent issue of hardware damage due to a controller computer connected to the printer going into sleep mode and causing significant hardware damage to the printer. At the time of design and throughout the development of the project, this issue was not known to the development team or the customer.

We have implemented several items to aid in the maintainability of this project:

- Modularity

We have strived to introduce modularity in the overall design, and within the individual functions contained in each module. The idea is that as better methods become available in the 3D printing community, or as fixes, changes, or improvements are made to the code base, those modifications will be relatively easy to make without disrupting unrelated portions of the code, and without breaking too many things.

- User manual

We have added a brief user manual section to this report to aid in the rudimentary use and installation of this software. This is a server side program running in a Linux environment, and as a result assumes some familiarity with Linux and system administration.

- Comments

We have made a conscious effort to thoroughly and purposefully comment the code to indicate what a module and its components do, and how they perform those functions.

Section 12: Conclusions

1. All but final pipeline stage work as planned
    - Can capture emailed model files and job originator info
    - Can convert submitted OBJ models to STL
    - Slic3r can validate and slice STL models

2. Cannot currently perform fully automated printing

    No single available program for communicating with the printer, that we have examined, has the right combination of open source code, compatibility with default manufacturer firmware, availability of a command line version, doesn't require installation of non-standard python libs, or no unusual and undocumented bugs that we need to successfully communicate with the printer without human interaction.

3. What we learned:
    - Even seemingly trivial things can turn out to be more complex than anticipated.
    - Writing a set of test cases before coding provides a good set of guidelines to code toward, but the fluid nature of development requires flexibility to revise, update, and improve those test cases.
    - We gained infinitely more respect for experienced project managers (in all engineering disciplines) who seen to have a knack for/know how to accurately estimate timelines, resources, schedules, budgets, etc… for successful projects.

4. What we would do differently:
    - Spend less time working with the Thing-o-matic printer, more time with target Replicator2 printer.
    - Time was a limiting factor, but it would have been nice to complete some of the stretch goals and enhancements that we have talked about. This is especially true for security enhancements, improved error handling, and in robustness and stress testing of the code.
    - We didn't consider the side effects of running on slow or congested wireless networks. Initial development was done with cable modem and gigabit ethernet.

5. Customer impression of the product:
    - Good, in spite of the fact that the final project stage is incomplete.
    - Interest is high with administrator at the Learning Center. He would like to extend the automated pipeline idea to download, validate, and slice e-mailed 3D models, and transfer them to CNC/machining processes.

Section 13: References

1. https://store.makerbot.com/replicator-mini
2. https://store.makerbot.com/replicator2
3. http://www.collexion.net
4. http://www.fcps.net/schools/others/the-learning-center
5. http://en.wikipedia.org/wiki/COCOMO
6. http://edutechwiki.unige.ch/en/Slicers_and_user_interfaces_for_3D_printers
7. http://reprap.org/wiki/G-code

Section 14: User's Manual and Installation Guide

Dependencies

Install dependencies meshlab, slic3r, and GPX.
Copy code directory or clone the repository onto your system.
Set up configuration information:

1. open mailfetch.conf
2. change Path under [Slicer] to the absolute path of the executable to Slic3r
3. change Path under [Converter] to the absolute path of the executable of meshlab
4. change Path under [Printer] to the absolute path of the executable of gpx
5. change server and username under Mailfetch to point to the email account you will be listening to.
6. change server and sender under Mailsend to reflect the account that you will be sending error messages and status reports from. This can be/most likely will be the same as the one under Mailfetch.

Plug in your printer via usb. GPX should automatically detect the device.

At this point, the application should be ready to use. There are two scripts that you can run. The first is main.py. This will ask you to log into the email account it will listen to, and then run as a daemon. It will attempt to 3d print any models emailed to the account specified in the [Mailfetch] section of mailfetch.conf. The usage is
python main.py
The other program you can use is mockmain.py. This will execute the pipeline on a local file instead of downloading it from email. This is useful if you already have a model from somewhere on a USB drive, and want to print it. The pipeline will greatly simplify the build process, even without reading from email. The usage is
python mockmain.py $MODELFILE