HÁSKÓLINN Í REYKJAVÍK
REYKJAVIK UNIVERSITY

T-404-LOKA-2015

Delivery Automation Tool for Isavia

Árni Þorvaldsson
Freyr Bergsteinsson
Gunnar Þór Helgason
Sigurbjörn Kristjánsson
BSc Computer Science 2015

Authors:
Árni Þorvaldsson 270874-3209
Freyr Bergsteinsson 010975-3019
Gunnar Þór Helgason 231074-4579
Sigurbjörn Kristjánsson 170488-2889

Instructor:
Daníel Máni Jónsson
Examiner:
Hannes Pétursson

Contents

Introduction

The Company

Tern Systems is an Icelandic company that reaches back to the late 1970's, but was formally established in 1997 by the University of Iceland and the Icelandic Aviation Administration. Tern Systems makes software for air-traffic control that is being used all around the world. Tern Systems is also maintaining and developing software for Isavia, its parent company. Isavia handles the operation and development of all airports in Iceland and manages air traffic in the Icelandic control area.

The official language at Tern Systems is English and thus all reports and documents regarding the project will be in English.

The Product

Software delivery for software maintained by Isavia often proves to be a tedious task sometimes requiring a developer to spend several hours manually checking to see if everything is in order. The product created by this project is called the Delivery Automation Tool - DATI for short - and it aims to solve that problem by automating the delivery process. For clarification, "the product" is what is being developed for Tern Systems, while "the project" is the graduate project at Reykjavík University.

DATI runs on any modern Linux distribution. The Operational Manual contains instructions on how to set up an operating system and all needed system components to run DATI.

To use DATI, only one command is needed:

```
$ scripts/create_delivery.sh <product> <version>
```

where <product> is the name of the product to create a delivery for, e.g. ISDS or ICE, which are actual product names at Tern Systems and <version> is the version of the product to create a delivery for, e.g. 14.05 or 15.02.

The User Guide contains further instructions on how to use and configure DATI.

1. Project Description

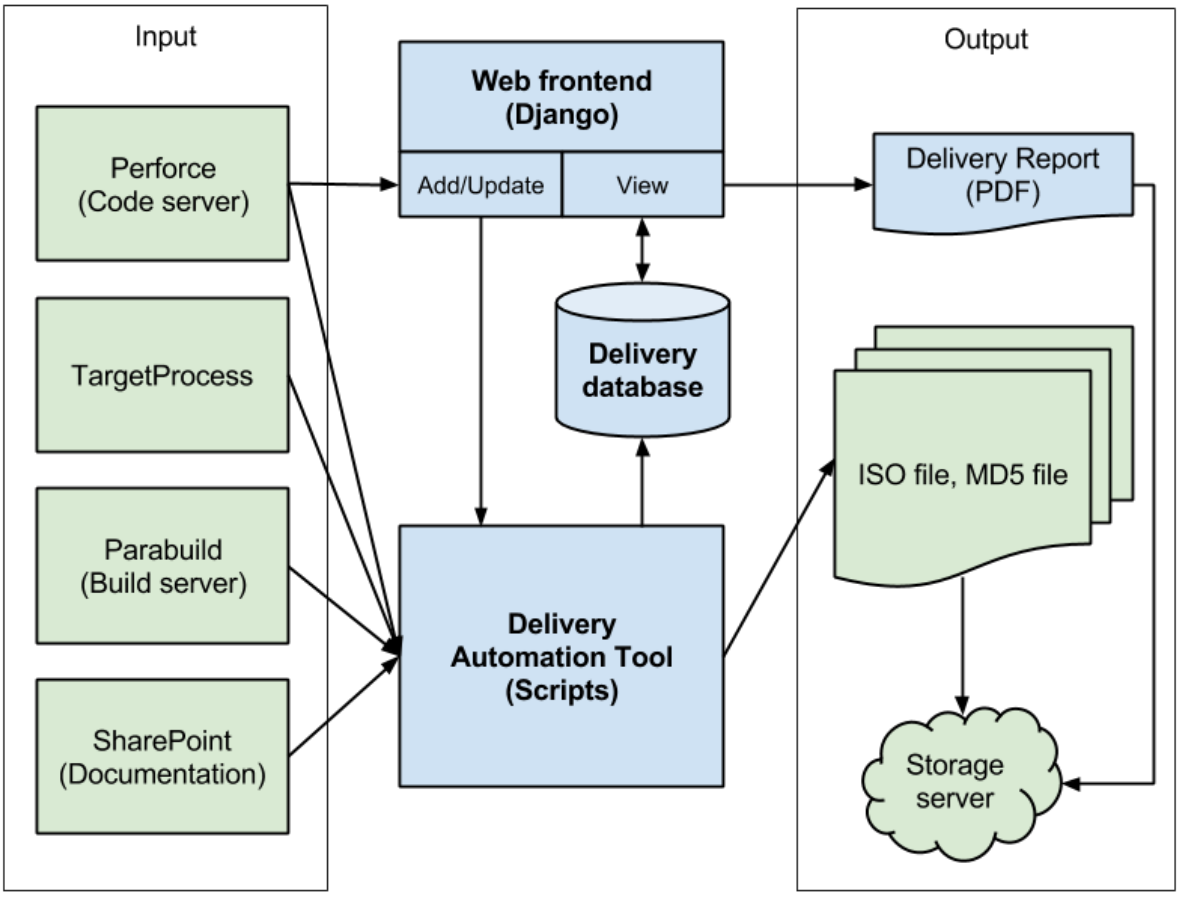

Figure 1. Overview of the system architecture. It shows the input from various systems into DATI. Pre-existing systems are depicted in green. DATI takes the information and produces a package ready for delivery to Isavia.

1.1. Description

The process that employees of Tern Systems go through when delivering software to Isavia has proven to be very time consuming and error prone. The process involves collecting and certifying great quantities of information. To this day this has been done manually. Going through this process can take two employees up to two days.

The project was to develop an automatic delivery software for Tern Systems that makes it easier and faster for employees of the company to deliver software to Isavia. DATI is a software that makes deliveries by collecting information via communication with Perforce, Parabuild, SharePoint and TargetProcess servers, all of which are vital to Tern Systems' development lifecycle. DATI gathers all this information into a package of an ISO file, a MD5 file and a PDF file which are then delivered to Isavia.

DATI is a command line tool and includes a collection of scripts that run in a specific sequence for a specific product delivery. It can also run one script at time to collect desired data.

The Django framework was used as a framework for DATI. Within Django the user can keep track of everything that is configurable like paths to certain system that needs to be accessed, the username and password for that specific system access. DATI version 1.0 comes with two views, an overview of stored deliveries and a report view for each individual delivery containing more detailed information about that delivery.

DATI communicates with 4 different systems to gather the data needed to complete the delivery. These systems are *Parabuild*, a continuous integration server, *SharePoint* for storage of documentation, *TargetProcess*, which is a project management system and *Perforce*, a revision control system.

Using DATI, the delivery process today is as simple as typing in one line at the command line and has shortened the time down to about 30-60 seconds. See chapter 4.3 for more detailed information.

1.2. Django Models

A database can be generated from Django models. For development of DATI, the following models were created; Products, ProductVersions, Scripts, ProductScripts, Config and Deliveries.

Products and ProductVersions store the names of products available for delivery and their version numbers respectively. The Scripts model stores the names of each script and contains the static function execute() that runs the corresponding script after setting up logging handlers. The ProductScripts model pairs scripts with products along with an assigned sequence number.

The two models that are most used are the Deliveries model and the Config. The former represents a delivery report and is used for storage of information for each individual delivery and is eventually used to generate the delivery report. The Config on the other hand works as a dictionary, with entries mapped to a string. The model stores all usernames, passwords (encrypted), URLs, folder paths, placeholder texts and product specific configurations. Adding new configurations is also easy through the Django admin site.

1.3 Script Overview

The following table contains an overview of the scripts that make up DATI. All scripts take a delivery number as their parameter unless noted otherwise. The scripts retrieve paths, usernames and other values from the config for use. Scripts exit with a status code 0 unless a fatal error occurs, in that case the script in question exits with a status code 1. The scripts are categorized by which systems they rely on, if any.

TargetProcess	
get_tasks_from_targetprocess.py	Accesses the TargetProcess API and retrieves a list of task numbers that is associated with the given product and version. Puts the task numbers along with description into the delivery object.
Perforce	
update_product_version.py	Checks Perforce which products are available for delivery and updates the database to reflect that. Accepts no parameters.
check_task_header_incremental_nr.py	Goes through all the files that have been modified since the last delivery for the product being delivered. For each file, it examines the task headers. Each task header entry has an entry number that starts with 1 and increments by 1 for each task header entry. This script makes sure that the task header entries conform to this.
check_task_header_dates.py	Goes through all the files that have been modified since the last delivery for the product being delivered. Script examines task headers in each file, making sure that the date format in each task header entry is the same within that file. The script also makes sure that the dates in the task headers that are new since the last delivery are are in range between the last delivery and the current day.
check_task_header_valid_tasks.py	Extracts task numbers from delivery object. Goes through new and modified files to confirm that new task header entries contain valid tasks. Raises warnings if either an invalid task is found or a valid one isn't found.

create_new_mod_del_files.py	Each delivery contains 4 files, one which lists which files are new since the last delivery, one which lists files are modified, one which lists files are removed and one file that contains the difference in the source files since the last delivery. The lists of new, modified and removed files are stored in the delivery object in corresponding fields.
Parabuild	
get_build_from_parabuild.py	Script retrieves the build that the Parabuild server has generated for the product and version that is being delivered. The script creates a SOAP client and uses the Parabuild web services API to find the relevant build result. Tarball(s) are then kept locally in a staging directory until later when the delivery is finalized.
SharePoint	
get_documentation_from_sharepoint.py	This script retrieves the documentation that the SharePoint server is storing for the product and version that is being delivered. The script then creates a REST client and uses the SharePoint web services API to find the relevant documentation file(s) by matching the product name and version in the delivery. The documentation files are kept locally in a staging directory until later when the delivery is finalized.
Independent scripts	
create_delivery.py	Takes 2 parameters, product and version and creates a corresponding delivery with an incremented delivery number. If a delivery already exists for the product and version given, the script returns the number of that delivery.
run_all_delivery_scripts.py	Runs all scripts for a given delivery. Order determined by configurations. Is called by create_delivery.sh along with create_delivery.py to make delivery from scratch.

task_header_parser.py	The task header parser finds task headers in files and creates task header entry objects for each task header entry within the task header. Each entry contains data about the task; date, description, author, entry number and task number along with a bool flag declaring whether the entry is new or not.
create_iso_and_md5.py	After all the items of the delivery have been put together in a staging directory, this script creates an ISO file from the contents of that directory and stores the ISO file in a delivery directory. A checksum file is then created for the ISO file, which is also stored in the delivery directory.
export_delivery_report_to_pdf.py	After all the items of the delivery have been put together, this script creates a PDF file from the view of the delivery in Django and stores it in the delivery directory. The PDF is created with a Python module called easy_pdf that relies on xhtml2pdf and reportlab.
copy_delivery_files_to_storage_server.py	When all delivery files are ready, this script copies them to a server for storage/backup. The files are uploaded using scp.
create_delivery.sh	Serves as a “master script”, takes in 2 parameters. product and version and runs the system with a series of scripts. The script run and their order is configured within Scripts module for each product.

1.4. Testing

Because Tern Systems specializes in making software for air traffic control, testing is an integral part of their development process. Strict testing requirements applied for this project as well and the team utilized Test Driven Development. It turned out that it was not possible in all cases to write unit tests before coding, but it was a rule that was followed as rigidly as possible. The testing process improved gradually over the span of the project and required several code refactorings. Mock objects were used for the unit tests and a close eye was kept on the percentage of code covering.

The project consists of 5981 lines of Python code (the unit tests alone make up 2556 lines of code) and 105 lines of bash code. As can be seen from appendix A, the Python code contains 3689 statements with a code coverage of 91%.

Information about how to create and run unit tests in DATI, as well as how to generate a code coverage report, can be found in the Operational Manual.

2. Working Arrangements

2.1. Methodology - Scrum

At Tern Systems work is done in accordance with Agile methodology, and both Scrum and Kanban are used. For this project, Scrum was our preference.

Team meetings were held at Tern Systems premises and daily meetings were held on Google Hangout the days that the team did not meet (except for weekends).

Sprints started and ended on a Monday with sprint reviews (including a demo), retrospective analysis and planning during the meetings. The duration of the introductory first sprint and the last 3 sprints, when team members had more time available, was one week, while the duration for the remaining sprints were 2 weeks.

Table 1. Overview of sprints

Sprint #	Sprint Name	Start Date	End Date
0	N/A	2015-01-19	2015-01-26
1	Grimlock	2015-01-26	2015-02-09
2	Bumblebee	2015-02-09	2015-02-23
3	Megatron	2015-02-23	2015-03-09
4	Ironhide	2015-03-09	2015-03-23
5	Jazz	2015-03-24	2015-04-05
6	Optimus Prime	2015-04-08	2015-04-19
7	Hot Rod	2015-04-27	2015-05-03
8	Mirage	2015-05-04	2015-05-10
9	Ultra Magnus	2015-05-11	2015-05-17

The user stories' story points were estimated with Planning Poker before Sprint #1 began. During all subsequent sprint planning meetings, the backlog was reviewed and updated according to the team's experience with story point estimation and the velocity of the team (backlog grooming). Planning Poker was then used for all new user stories.

When planning the sprints, user stories were prioritized according to what's called MoSCoW ("Must have, Should have, Could have, Won't have"). Then each user story was broken down into as small tasks possible and the remaining work (in hours) for each task was estimated. At the start of the project Trello was used to keep track of the Scrum board and product backlog, but it was soon replaced with Visual Studio Online, which was better suited to the requirements of the team. User stories that were selected for a sprint were removed from the backlog and assigned accordingly.

Sprint reviews and retrospectives were held after each sprint. Retrospectives were held immediately after status meetings with examiners and instructors as well so issues raised at the meetings could be noted down at once.

2.2. Roles and Responsibilities

2.2.1. Team

The Autobots team consists of the following members:

- Árni Þorvaldsson
- Freyr Bergsteinsson
- Gunnar Þór Helgason
- Sigurbjörn Kristjánsson

Each member has the role of a developer, which is an interdisciplinary role, meaning everyone takes on tasks that pertain to coding, documenting, testing and more.

2.2.2. Product Owner

The Product Owner is Heiðar Harðarson, which is also the team's correspondent at Tern Systems. Mr. Harðarson has been employed by Tern Systems since 2000 as a programmer and project manager. He leads the team at Tern Systems that is responsible for development and maintenance of systems owned by Isavia. This gives him a very clear vision in regards to what the end-product of this project should feature.

2.2.3. Scrum Master

The Scrum Master of the team is Freyr Bergsteinsson. It was decided he would take this role because of previous Agile experience, even though he has more experience with Kanban than with Scrum. In addition to being a student, Freyr is also an employee at Tern Systems.

2.3. Software Environment

The project is made mostly of Python and Bash files and text editors and PyCharm were used for coding. The code was kept on a private repository on github. Django is used to configure the system and for the web-based front end. The database system chosen for the project is MySQL.

2.4 Time registration

A journal was shared on Google Drive to keep track of the group members working hours, with tables and live charts to show progress visually.

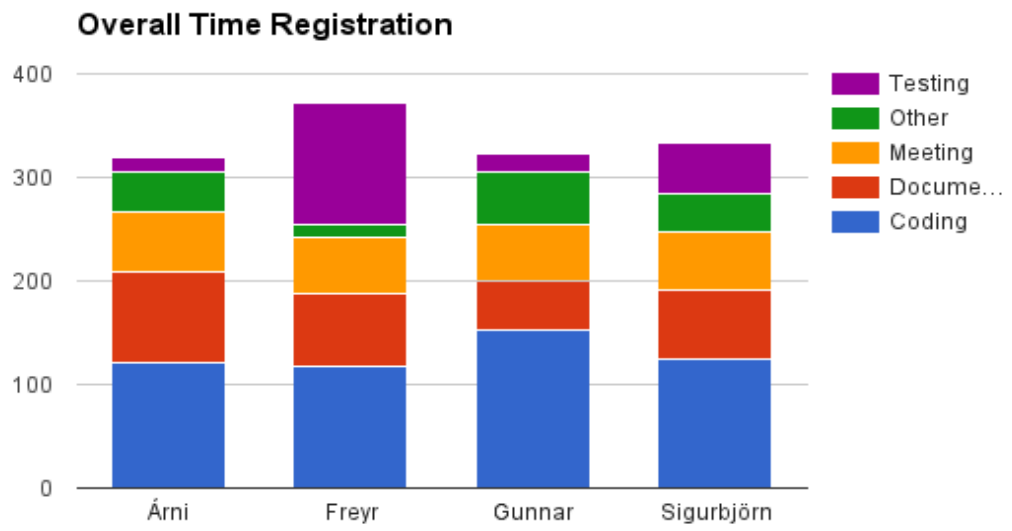

Figure 2. Overall time registration

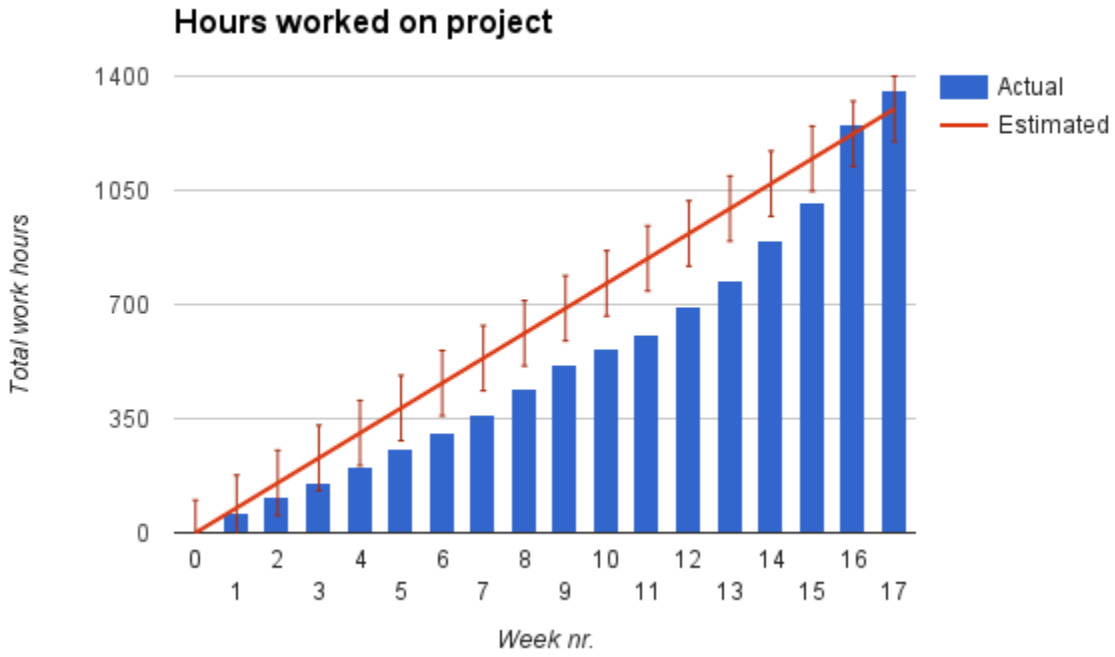

Figure 3. Summary of weekly work hours in relation to desired timeframe

3. Progress

3.1. Cumulative Flow Chart for User Stories

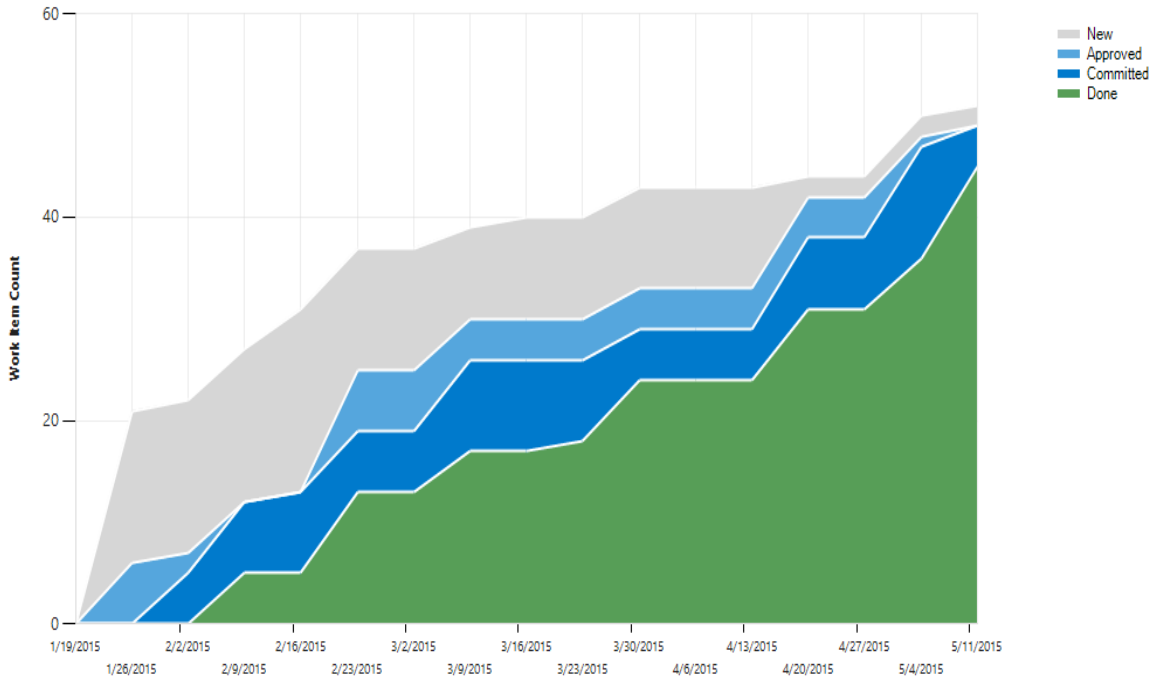

Figure 4. Cumulative flow chart for user stories.

As can be seen from the cumulative flow chart, most of the new user stories were drawn up in the first month of the project when the emphasis was on design and planning. Then at the end, when the project is reaching its conclusion, there is a new peak when unforeseen user stories were added.

3.2. Sprint 0

Sprint 0 was one week long and was mostly seen as an introduction to the project and for choosing and setting up software and tools. The team agreed on using the Scrum software development framework for managing product development. First draft of the project report was completed.

3.3. Sprint 1

First hand-in of reports to examiner. Developer server and web server were set up. The following user stories were selected for the sprint. One story (“As a user I want to configure the tool to run appropriate tasks”) was removed after the start of the sprint due to updated requirements. Another story (“As a user I want to make a master script that runs the whole project so that it is simpler to create delivery”) was left unfinished and moved to next sprint.

Table 2. User Stories in sprint 1

User story	Est. story points
As a user I want to configure the tool to run appropriate tasks	8
As a user I want to make a master script that runs the whole project so that it is simpler to create delivery	2
As an instructor and examiner I want to see the team's working arrangements so I can assess the progress of the project	1
As an instructor and examiner I want to see the team's Project Schedule so I can assess the progress of the project	1
As an instructor and examiner I want to see a presentation of the product so I can assess the progress of the project	1
As as user I want a web front end to the product so that I can interact with it	2
As a developer I want a server for the product so that I can run a centralized server for web front end and running of scripts	2

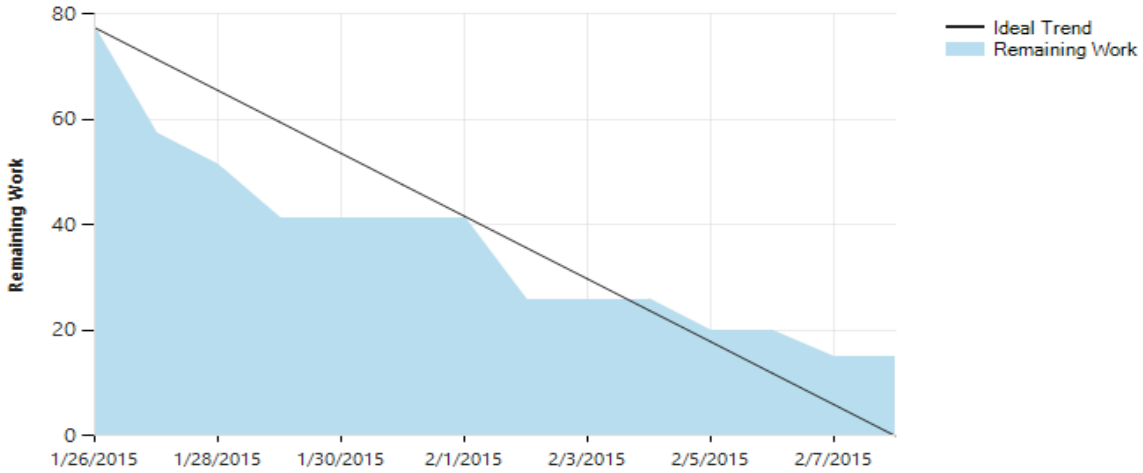

Figure 5. Sprint 1 burndown chart.

Points from sprint retrospective:

What went well:

- Work on reports went well.
- Preparation for presentation of project went well.

What could be improved:

- Draw diagrams of user stories and tasks to enhance understanding of the project in a visual way before coding is started. Will also be of use for instructor and examiner.

3.4. Sprint 2

The emphasis was on design and reports. First steps were taken in coding and the first draft of a master scripts was completed.

Table 3. User stories in sprint 2

User story	Est. story points
As a user I want to make a master script that runs the whole project so that it is simpler to create delivery	5
As an instructor and examiner I want to see the team's Developments Report so I can assess the progress of the project	2
As an instructor and examiner I want to see the team's Risk Analysis so I can assess the progress of the project	2
As an instructor and examiner I want to see the team's updated Project Schedule so I can assess the progress of the project	1
As a developer I want a better overview of the internal working of the product so that I better understand it	3
As a scrum master I want to see scrum tools in order so that we can report work progress (Framvinduskýrsla)	1

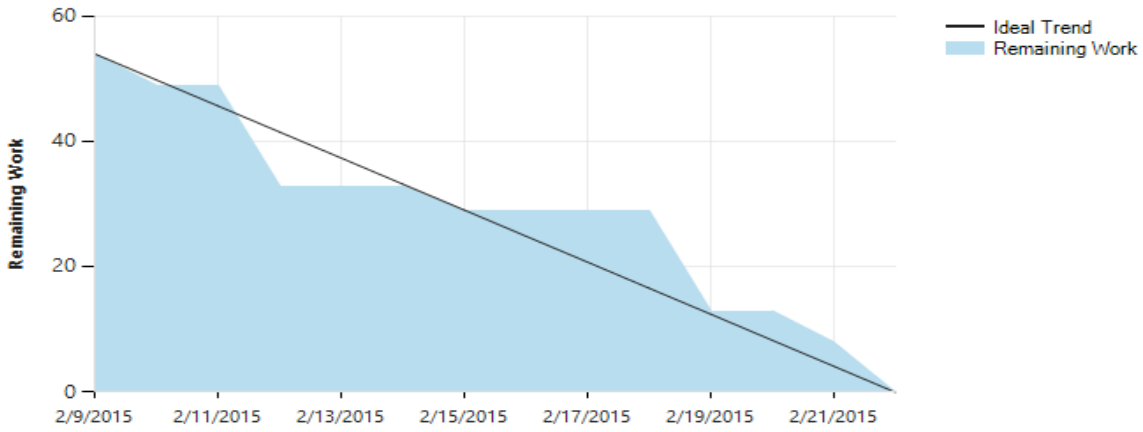

Figure 6. Sprint 2 burndown chart.

Points from sprint 2 and status meeting 1 retrospectives:

What went well:

- Better workflow than in previous sprint
- Status meeting 1 very informative
- Good spirits within team

What could be done better

- Explain product better to examiner
- Team members should be more vigilant in asking for help
- Document what to do in more detail before coding
- Work journal entries disappeared
- Further highlight the dangers and pitfalls in the project
- In risk analysis don't make the risks too general

Actions

- Update documentation with points from status meeting
- Update work journal summary to better handle entries, preferably so that entries can be assigned to sprints by date alone

3.5. Sprint 3

A good deal of the time in this sprint was spent on unit tests. Work began on creating a design document, which would prove very useful in the forthcoming sprints when work on scripts started. Two stories were left unfinished in the sprint and moved to next sprint.

Table 4. User stories for sprint 3

User story	Est. story points
As a developer I want to easily add unit tests to the product so that the product is more reliable	2
As a developer I want the config manager in a separate database than the delivery report as to keep the tool separate from the data	2
As a developer I want the Config Manager to be part of the Django framework so that it is easier to maintain	4
As an instructor and examiner I want to see the project documents updated with regards to points made in Status Meeting 1	1
As a user I want delivery reports to be stored in a database so I can use it to generate a report later	2
As an instructor and examiner I want to see a draft of a design document so that I can better understand how the product is supposed to work	3

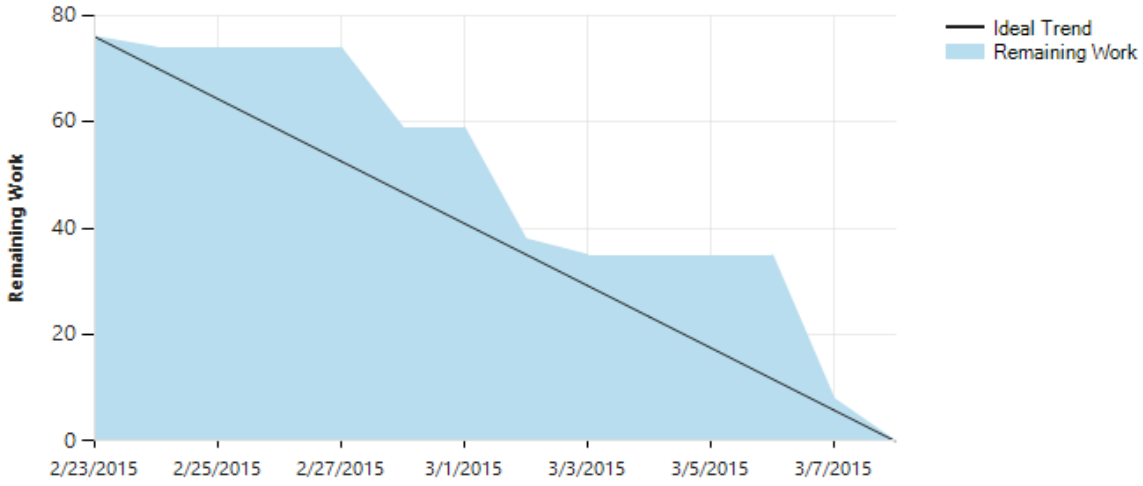

Figure 7. Sprint 3 burndown chart.

Points from sprint 3 retrospective :

What went well

- Better overview of the project than before
- Unit tests went well
- Diagrams help to visualize and deepen understanding

What could be done better

- Encourage dialogue between group members - ask questions

Actions

- Setup virtual environment?
- Setup Django locally on all computers

3.6. Sprint 4

After the three first sprints the team could assess its velocity quite accurately, which would hold up well in the following sprints. Having established an accurate velocity also facilitated the subsequent sprint plannings.

From this point on, when most of the groundwork had been laid regarding design and planning, the emphasis in the project shifted towards writing scripts and adding unit tests.

Table 5. User stories for sprint 4

User story	Est. story points
As a user I want to be able to look at and edit data in an admin web interface to better control the data in a visual manner	1
As a user I want delivery reports to be stored in a database so I can use it to generate a report later	2
As an instructor and examiner I want to see a draft of a design document so that I can better understand how the product is supposed to work	3
As a user I want to fetch a list of projects and versions from Perforce so I can use them to select what to deliver	3
As a developer I want to have a list of auxiliary requirements so that I can continue developing if all other requirements are met	1
As an instructor and examiner I want to see the work progress updated regularly to have a better overview of the project	1
As an instructor and examiner I want to see a presentation of the product for the second status meeting so I can assess the progress of the project	1
As a user I want to automatically get user manuals for a specific version of my software from SharePoint so I can add it to the delivery	3

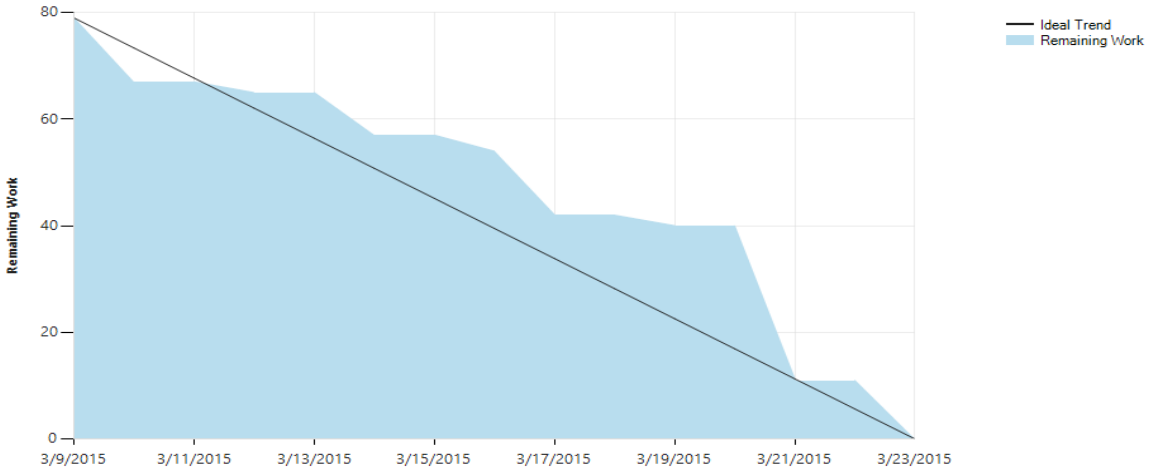

Figure 8. Sprint 4 burndown chart.

Points from retrospectives after sprint 4 and status meeting 2

What went well

- Generally positive status meeting
- Few questions, instructors seemed to get the point of the project
- We got something to run (which shows some functionality)

What could be done better

- Explain which scripts are completed and what is left to-do
- Attendance at daily meetings

Actions

- Deal with request time-out from browser
- Use RabbitMQ?

3.7. Sprint 5

The greatest challenge in this sprint was to write the script that connects to Parabuild using SOAP API, retrieves data and stores it in a storage server.

Table 6. User stories for sprint 5

User story	Est. story points
As a user I want to be able to fetch a build from Parabuild so that I can include it in the delivery	5
As a user I want the delivery to show up in a list of deliveries (in Django) so I can see what has been delivered	2
As a user I want the delivery output files to be uploaded to a storage server so that I have a backup of the data	3
As an instructor and examiner I want to see the work progress updated regularly to have a better overview of the project	1
As an instructor and examiner I want to see the project documents updated with regards to points made in Status Meeting 2	1

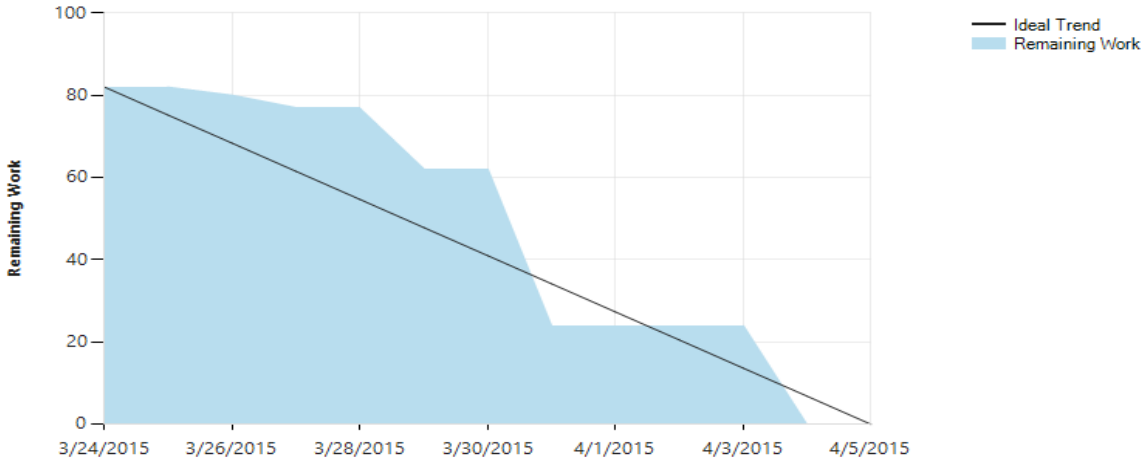

Figure 9. Sprint 5 burndown chart.

Points from sprint 5 retrospective:

Actions

- Check timeout for larger files when copying to storage server
- Add to Risk Analysis risk of cloud based programs
- Explain to instructors about low priority of web front end

3.8. Sprint 6

For this sprint the following user stories were selected:

Table 7. User stories for sprint 6.

User story	Est. story points
As an instructor and examiner I want to see the work progress updated regularly to have a better overview of the project	1
As a developer I want a parsing tool to recognize task headers so that it becomes easier to examine them in scripts	5
As a developer I want to connect to TargetProcess through an API to get access to task numbers	4
As a user I want to make sure that task header numbers are correctly numbered in an incremental fashion so that the delivery isn't rejected	2

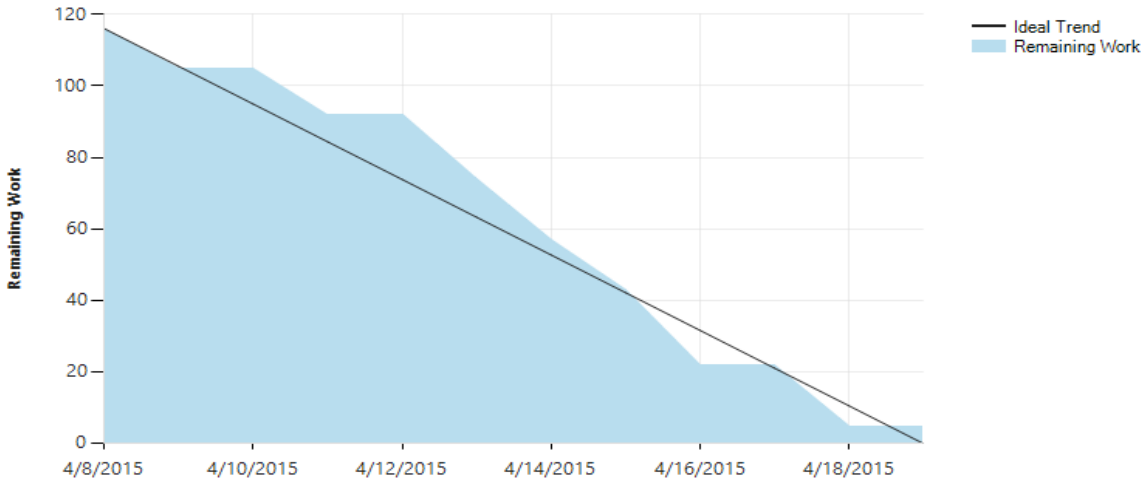

Figure 10. Sprint 6 burndown chart.

Points from sprint 6 retrospective:

What went well

- Work on scripts went well

What could be done better

- Less capacity because of exams was not taken into account in sprint planning

Actions

- Update copy_delivery_to_storage_server so it does not overwrite existing
- Sharepoint config - use folder instead of an url to retrieve documents

April 21st - April 26th: The team took 6 days time off due to exams and family obligations before the start of sprint 7.

3.9. Sprint 7

Sprint length was reduced from two weeks to one week for sprint 7 and the following sprints, as other school courses were finished and team members could henceforth focus solely on the project. Thus team capacity and velocity estimations were increased for this sprint and the following.

Table 8. User stories for sprint 7

User story	Est. story points
As an instructor and examiner I want to see the work progress updated regularly to have a better overview of the project	1
As a user I want to make sure that the task headers delivered contain valid task numbers so that the delivery isn't rejected	3
As a user I want task headers to show the correct date in task headers so that the delivery isn't rejected	5
As a user I want to retrieve all documents from a folder in SharePoint for a product instead of one document	3
As a user I don't want old backup files to be overwritten in case of mistakes in new delivery	3
As an instructor and examiner I want to see a presentation of the product for the third status meeting so I can assess the progress of the project	1
As a user I want to be able to create a new delivery with a command line script or perform re-delivery on an existing delivery	3

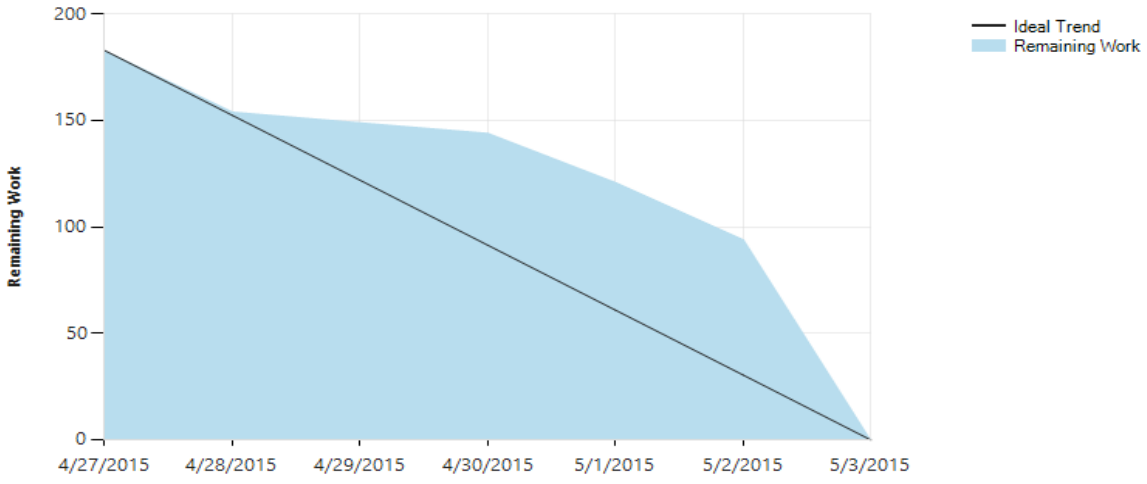

Figure 11. Sprint 7 burndown chart.

Points from sprint 7 retrospective:

What could be done better

- Unit tests were very challenging in this sprint

Actions

- Make use of collaboration when pair coding

3.10. Sprint 8

As this was supposed to be the last sprint with coding, an exceptionally high number of story points were chosen for the sprint, and as can be seen from the burndown chart, not everyone of them could be finished.

Table 9. User stories for sprint 8

User story	Est. story points
As an instructor and examiner I want to see the work progress updated regularly to have a better overview of the project	1
As a user I want to make sure that the task headers delivered contain valid task numbers so that the delivery isn't rejected	3
As an instructor and examiner I want to see a presentation of the product for the third status meeting so I can assess the progress of the project	1
As a user I want to view one specific delivery report so I can verify its correctness	3
As a user I want to generate a delivery report in PDF format so that I can add it to the delivery	3
As an instructor and examiner I want to see the project documents updated with regards to points made in Status Meeting 3	1
As a user I want the run_script script to behave the same as the Script.execute method	1
As a user I don't need config values for a specific product version (only for products)	1
As a user I want the system to work on product versions regardless of minor version being zero padded or not	3
As a user I want the system to automatically get a list of new, modified and removed files between to deliveries	4
As a user I want the system to automatically create an ISO of a staging directory and MD5	3

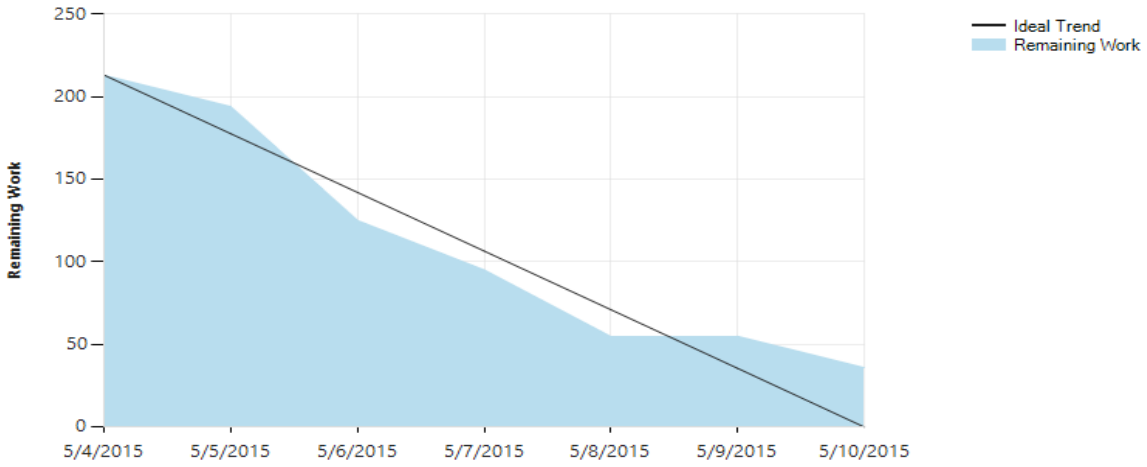

Figure 12. Sprint 8 burndown chart.

Points from sprint 8 retrospective:

Actions:

- Deployment of DATI
  - from runserver(Django) to Apache server
- Demo failure
  - new delivery does not work - missing tern_base_tag and tern_delivery_tag
- For final presentation:
  - show the sequence diagram for each script when running demo
- Finish info log for all scripts

3.11. Sprint 9

In the last sprint emphasis was on documentation; updating reports, creating a final report and a final presentation. Some work was done on reviewing, refining and testing code as well. The sprint was still on-going at the time of the final hand-in.

Table 10. User stories for sprint 9.

User story	Est. story points
As an instructor and examiner I want to see the work progress updated regularly to have a better overview of the project	1
As an instructor and examiner I want to see the project documents updated with regards to points made in Status Meeting 3 (unfinished from previous sprint)	1
As an instructor and examiner I want to see a final report for the project so I can grade the project	8
As an instructor and examiner I want to see a presentation of the product for the final presentation	3

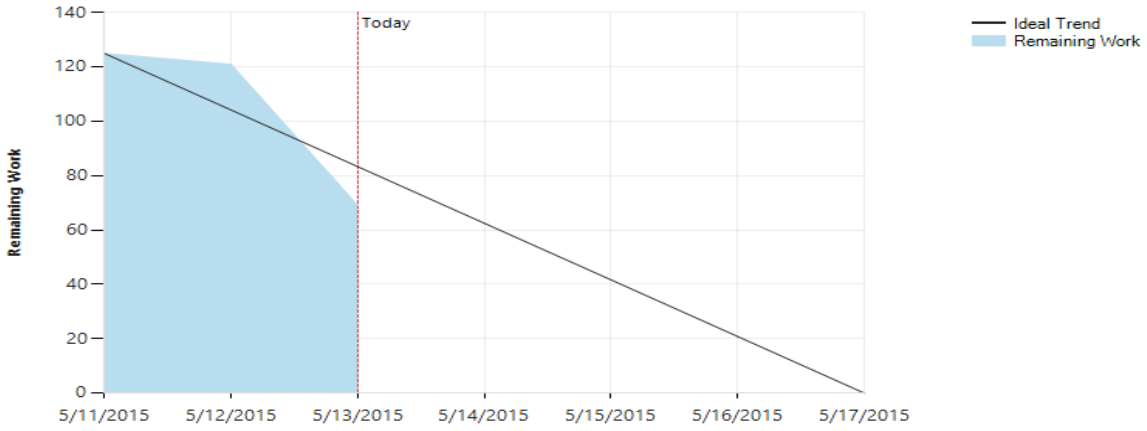

Figure 13. Sprint 9 burndown chart.

4. Summary

4.1 Team Experience

DATI is a flexible tool and it its easy to implement new scripts and features. It could be adapted to other systems than it was especially developed for with minor revisions. DATI will be implemented in Tern Systems by the team which faces the delivery process containing the highest complexity. In the future, DATI will likely be used by other teams at Tern Systems as well. DATI is primarily a command line tool but adding a web based front end is an option. The team has a clear view of how the web front end could look in the future, with a delivery creation mode and visual picking scripts that benefit a specific product.

The Autobots Team members agreed that the work on the project was very instructive. The challenges in the project were many. The majority of team had little or no experience with Python and none of us had ever used Django before. Interaction with other systems like Perforce, SharePoint, Parabuild and TargetProcess through APIs was a big challenge for the majority of the team members, who had never done that kind of work before.

Although we all had previous experience of using Scrum it was still a challenge to run it smoothly. In retrospect the Scrum methodology turned out to be an invaluable tool. A few points that we learned from the process:

- The importance of breaking user stories/tasks into as small units as possible.
- The retrospectives are very important and offer a built-in continuous process improvement.
- The importance of writing good stories, specifying especially the who, what and why of each feature.
- Tracking team velocity greatly helps for further sprint planning.

There are strict testing requirements at Tern Systems and those applied for our project as well. To write unit tests and mocks in Python was a good learning experience and the good code

coverage percentage fills us with confidence that we've created a system that will become useful for Tern Systems and will soon replace the time-consuming and error-prone process that has been in use for years.

4.2 User Experience

A few users were interviewed about their experience with the old delivery procedure with regards to time spent performing them.

Table 11. User experience

			
Name	Sigurjón Örn Sigurjónsson	Birgir Ragnarsson	Davíð Halldór Kristjánsson
Job Title	Software Engineer	Software Engineer	Software Engineer
Division	System Solutions - ACC	System Solutions - ACC	System Solutions - Simulator
Comments	I spend anywhere from 3 to 16 hours on the delivery. I estimate it takes 6 hours on average.	Delivery takes 4 to 24 hours, depending on whether there are errors in the delivery. I spend 7 hours on average for each delivery.	Our team does not deal with task headers, but an official delivery still takes 4 to 8 hours.

4.3 Results and Comparison

Table 12 shows data provided by Tern Systems directly from their time registration system (which is used to bill Isavia) for the last three deliveries made.

Table 12. Time spent on delivery

Delivery Name	Delivery Date	Hours billed
ISDS 15.01	13.03.2015 12:00	27.50 hrs
ISDS 15.02	15.04.2015 15:30	17.50 hrs
ISDS 15.03	11.05.2015 15:00	11.25 hrs

Table 13 shows data of how long DATI took to recreate the same deliveries as in Table 12. Each delivery was created three times and the average running time is shown.

Table 13. Running time of DATI

Delivery Name	Running time
ISDS 15.01	36.146 sec
ISDS 15.02	N/A
ISDS 15.03	31.321 sec

DATI was unable to recreate the ISDS 15.02 delivery because TargetProcess contained insufficient data. This is due to the fact that Tern Systems recently changed from JIRA to TargetProcess and are still in the process of migrating data between the two systems at the time of this writing.

In addition to running DATI, a human would have to take time to set up the delivery process, verify that the output is correct and possibly respond to any warning messages produced by DATI. It is fair to assume that, even with DATI, the new delivery process will still take 0.5 to 3 hours, which means we are reducing time spent on the deliveries in Table 12 by 88% to 98%.

4.4 Review from Contact / Product Owner

The following is a direct comment made by our contact and product owner at Tern Systems, Heiðar Harðarson, in Icelandic:

> “Viðfangsefni verkefnisins er sjálfvirknivæðing **vinnufreks** og **villugjarns** vinnuferils við að **afhenda** hugbúnaðaruppfærslur til okkar stærsta viðskiptavinar. Afurð verkefnisins er **afhendingin**, þ.e. það sem við afhendum viðskiptavininum. **Afhendingin** er í grunninn byggð á afhendingu kóðaskráa og handbóka ásamt greinargerðum um hverju var breytt og afhverju. Viðskiptavinurinn byggir síðan kerfið sitt sjálfur og setur í rekstur.
>
> Þessi vinna er oftast allt of tímafrek, betra og skemmtilegra væri að nýta tímann í skapandi vinnu við verkefnin sjálf. Og villur rata stundum inn í skilavinnunna þannig að hún sem slík stenst ekki væntingar viðskiptavinarins, sem kallar þá á endurtekningu með lagfæringum á skilavinnunni sjálfri - ekki vörunni.
>
> Hópurinn hefur greint þarfir verkefnisins í þaula og afurðin lofar mjög góðu. Skilavinnan mun væntanlega minnka niður í hluta úr degi. Ég vænti þess að við tökum hana í notkun fljótlega með ávinningi fyrir okkur og viðskiptavininn.”
>
> - Heiðar Harðarson, Project Manager at Tern Systems

Appendices

Appendix A - Code Coverage

Table 14. code coverage.

Module	*statements*	*missing*	*coverage*
configmanager/__init__	0	0	100%
configmanager/admin	6	1	83%
configmanager/models	66	0	100%
configmanager/test_config	148	0	100%
configmanager/views	1	1	0%
dati/__init__	0	0	100%
dati/settings	19	0	100%
dati/urls	6	6	0%
dati/wsgi	4	4	0%
deliveries/__init__	0	0	100%
deliveries/admin	3	0	100%
deliveries/models	61	0	100%
deliveries/tests	71	0	100%
deliveries/urls	5	5	0%
deliveries/views	22	22	0%
manage	6	0	100%
products/__init__	0	0	100%
products/admin	4	0	100%
products/models	46	0	100%
products/test_product	69	0	100%
products/test_product_version	51	0	100%
products/views	1	1	0%
scripts/__init__	0	0	100%
scripts/admin	4	0	100%
scripts/collection/__init__	0	0	100%
scripts/collection/check_task_header_dates	94	0	100%
scripts/collection/check_task_header_incremental_nr	65	25	62%
scripts/collection/check_task_header_valid_tasks	94	9	90%
scripts/collection/copy_delivery_files_to_storage_server	87	9	90%

scripts/collection/create_delivery	67	9	87%
scripts/collection/create_iso_and_md5	53	2	96%
scripts/collection/create_new_mod_del_files	102	11	89%
scripts/collection/export_delivery_to_pdf	48	21	56%
scripts/collection/get_build_from_parabuild	144	22	85%
scripts/collection/get_documentation_from_sharepoint	137	78	43%
scripts/collection/get_tasks_from_targetprocess	71	35	51%
scripts/collection/run_all_delivery_scripts	37	37	0%
scripts/collection/update_product_versions	42	7	83%
scripts/messages	64	1	98%
scripts/mocks/__init__	0	0	100%
scripts/mocks/always_callable	14	2	86%
scripts/mocks/mock_parabuild	74	2	97%
scripts/mocks/mock_perforce	114	12	89%
scripts/mocks/mock_ssh	39	6	85%
scripts/models	117	3	97%
scripts/task_header_parser	148	12	92%
scripts/test_check_task_header_dates	136	0	100%
scripts/test_check_task_header_incremental_nr	71	0	100%
scripts/test_check_task_header_valid_tasks	109	0	100%
scripts/test_copy_delivery_files_to_storage_server	97	0	100%
scripts/test_create_delivery	21	0	100%
scripts/test_create_iso_and_md5	98	1	99%
scripts/test_create_new_mod_del_files	96	0	100%
scripts/test_export_delivery_to_pdf	32	0	100%
scripts/test_get_build_from_parabuild	113	0	100%
scripts/test_get_documentation_from_sharepoint	74	0	100%
scripts/test_get_tasks_from_targetprocess	49	0	100%
scripts/test_productscript	259	0	100%
scripts/test_script	104	0	100%
scripts/test_task_header_date_parser	15	0	100%
scripts/test_task_header_parser	180	0	100%
scripts/views	1	1	0%
scripts/wrapper	30	0	100%
Total	**3689**	**345**	**91%**

Appendix B - Terms

Table 15. Terms

English	Icelandic
Design Document	Hönnunarskýrsla
Development Report	Framvinduskýrsla
Operational Manual	Rekstrarhandbók
Project Report	Verkefnislýsing
Project Schedule	Verkáætlun
Risk Analysis	Áhættugreining
Status Meeting	Stöðufundur
User Guide	Notendaleiðbeiningar
Working Arrangements	Verkskipulag