New technologies for eXtremeManagement

Maurits van Rees
Student number 0545701
Hogeschool Rotterdam
February - June, 2007
New technologies for eXtremeManagement

Maurits van Rees
Student number 0545701
Class inf8
Hogeschool Rotterdam
February - June, 2007
Teachers:
Hans Manni and Aad van Raamt
Preface

This report describes my final assignment for my study Informatics at the Rotterdam Institute for Informatics Studies (RIVIO) of the Hogeschool Rotterdam. The assignment lasted from February till June 2007. It was carried out for web development company Zest Software in Hoogvliet, The Netherlands. I have been working there since November 2005. Previous reports for school of my work for Zest Software (including this report) can be found on my website\(^1\).

The main subject of this report is eXtremeManagement. This is a project management tool for the open source content management system Plone. At Zest Software we use this tool on a daily basis to keep track of what our customers want and how much time we have worked for them.

This eXtremeManagement tool can use some updates, which is the goal of this assignment. The focus is on the underlying technology: using more and more features made available by newer versions of Plone and the Zope web development framework that Plone is built on. Also some user interface improvements will be made.

I thank Hans Manni from RIVIO for keeping me on track for finishing my study. I thank Aad van Raamt for being the second teacher next to Hans on the committee. I thank Jean-Paul and Esther Ladage for giving me the opportunity to work on eXtremeManagement for five months. I thank Reinout for the photo\(^2\) on the front page. I thank my colleagues from all over the world for their feedback, their own additions to eXtremeManagement and for making Zest Software a very nice team to work and have fun with.

Rotterdam, June 2007
Maurits van Rees

\(^1\)http://maurits.vanrees.org/studie
\(^2\)http://flickr.com/photos/reinoutvanrees/492407522/
## Contents

1 Introduction 1
   1.1 Crash course ........................................ 1
   1.2 eXtremeManagement ................................. 2
   1.3 Structure ........................................... 3

2 User Interface 5
   2.1 Overviews ........................................... 5
   2.2 AJAX ................................................ 7

3 Plone 3.0 9
   3.1 General actions .................................. 9
   3.2 More fixes ......................................... 11

4 Zope 3 14
   4.1 Interfaces ......................................... 14
   4.2 Browser views ..................................... 16
   4.3 Utility .............................................. 21
   4.4 Annotations for time registration ................. 22

5 Conclusion 27

Dutch summary 28

A Future plans 29
Abstract

This report describes my final assignment for school, which is: improve the eXtreme-Management tool: a project management tool based on Extreme Programming principles and running on content management system Plone.

The Introduction (chapter 1) paints the landscape of this assignment. What is Zope? What is Plone? What is eXtremeManagement? How do they fit together?

I made some improvements to the User Interface (chapter 2). Most of the original ideas there were not implemented however, for various reasons, ranging from simple lack of time to fresh insights that invalidated the original plan. The focus of the assignment was shifted to improving the core, instead of the front door.

Plone 3.0 (chapter 3) tells the tale of getting eXtremeManagement ready to run on the new (still in beta status) version 3.0 of Plone. I did some standard fixes applicable to all third party Plone products. I also did some other changes that were found to be needed. Finally I added an improvement to core Plone to make this upgrade easier for other products.

With Zope 3 (chapter 4) we come to the heart of the matter. More than originally envisioned the focus needed to be put here. I added marker and functional interfaces. I created browser views to make a clearer Model-View-Controller distinction. I introduced annotations for keeping track of estimates and hours worked. All three work together to make a far cleaner version of eXtremeManagement than was there at the beginning of this assignment.

I draw the Conclusion (chapter 5) that eXtremeManagement is clean and future-proof and that I have learned a lot in this assignment.
Chapter 1

Introduction

1.1 Crash course

Most chapters of this report assume at least some knowledge of Plone and Zope and related technologies, as eXtremeManagement uses them. Readers who are not that familiar with these technologies may benefit from some introductory remarks.

eXtremeManagement Project management tool that runs on Plone. In daily use at Zest Software. It is the subject of this report.

Plone Content Management System, useful for websites that need easy, through-the-web editing and a lot of flexibility. Uses the CMF.

CMF Content Management Framework. It offers basic building blocks for Plone like QuickInstaller and GenericSetup: both tools to help configure and install Plone and other products. Runs on top of Zope.

Zope Web Application Framework, useful on its own or as a basis for systems like CMF. Written in Python. Zope comes in two versions.

Zope 2 This is the most used version. Plone needs this. Internally, however, more and more functionality is gradually taken over by Zope 3, which is shipped as part of Zope 2. That bears repeating: Zope 3 is now an integral part of Zope 2.

Zope 3 Next generation of Zope. The code of Zope 3 itself is far cleaner and saner than that of the previous version. Third party programs based on Zope 3 (or Zope 2 programs partly using Zope 3) also tend to be cleaner and better structured than pure Zope 2 applications. Plone is also using Zope 3 technologies more and more.
CHAPTER 1. INTRODUCTION

Python Dynamic, object-oriented programming language; Plone, Zope and most of the other programs mentioned in this report are written in this language.

collective Subversion (version control) repository for third party products for Plone. The eXtremeManagement code is available there.

What better way to show this than with a picture. On the left is the situation as it is used by eXtremeManagement. Python is the base, on top of that is Zope 2, which partly uses Zope 3, and on top of that run CMF, Plone and finally eXtremeManagement. The picture on the right is there to make clear that third party products can also build on Zope 3 directly, instead of relying on intermediate levels.

1.2 eXtremeManagement

Let’s get a clearer picture of what eXtremeManagement is. It is:

- a project management tool
- heavily influenced by the Extreme Programming methodology
• a Plone product, so it is available through the web as an online collaboration tool: programmers and clients from all over the world can login and look at their relevant projects.

A picture is worth a thousand words, so please take a look at the next screen shot. The underlying Zope technology stores objects in its object database. In most cases objects are instances of content types. The most important content types that eXtreme-Management adds, from high to low level, are:

**Project** This is basically a folder. It functions as the central place to store information about a project, especially Iterations.

**Iteration** A Project is split into several Iterations. An Iteration is a period of a few weeks during which work in the form of Stories is done.

**Story** A Story (also called user story) is a piece of functionality, for instance: “As an Employee I want to see my total hours per week.” This should be phrased in a way that both developers and clients can understand.

**Task** A Story is split into several Tasks: concrete packets of work for an employee, for instance programming some functionality or improving the lay-out of a website.

**Booking** A Booking is a measure of time: several hours and minutes. A Booking is added to a Task to make clear how much work has been done on that Task. Based on the total of Bookings in a complete Iteration, you can send a bill to the client.

### 1.3 Structure

Each of the following chapters and sections starts with an intro that shortly explains the original plan at the start of the assignment. The subsections then describe the real work that was done, followed by a subsection *Lessons learned* which is more about the process and what I learned from the subject of that section.
CHAPTER 1. INTRODUCTION

---

eXtreme Management

by admin — last modified 2006-02-08 19:59  [H] History

This project is here to maintain the eXtreme Management tool itself. A funny way of recursion.

Current Iterations

<table>
<thead>
<tr>
<th>release 1.2.7</th>
<th>50 hours (34:30 actual)</th>
<th>status: in-progress</th>
</tr>
</thead>
<tbody>
<tr>
<td>story</td>
<td>tasks open</td>
<td>tasks completed</td>
</tr>
<tr>
<td>[ ] Size estimates</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>[ ] More Plane 3 improvements</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>[ ] As an Employee I want to see my total hours per week</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>[ ] Project Management</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

---
Chapter 2

User Interface

2.1 Overviews

In the tool a lot of information is available. Not all information is readily visible though. The plan at the beginning of the assignment was to add at least two things and maybe some more.

2.1.1 Tasks per employee

Managers should be able to see which tasks an employee is assigned to. At the moment members can see their own tasks, but others can only see how many hours someone is assigned totally on a project.

This idea is probably quite doable, but is postponed for now. Currently before an iteration starts all tasks are assigned to employees. We are pondering a change there though. We may want to let employees sign up for a new task when they are done with the previous one. Then there would not be many tasks to show the manager in such an overview. Currently though, showing the number of tasks for everyone could be overwhelming.

So the question is: where do you show that overview and how do you show it? We have not got a clear picture on that yet, so it is postponed for the moment.

2.1.2 Bookings per week

Employees would like to see how many hours they have already worked this week or last week. We can do that. Let’s look at my bookings from the first half of May 2007. Look at the next two images to see how it used to look and how it looks now.
CHAPTER 2. USER INTERFACE

Bookings in May

Before:

<table>
<thead>
<tr>
<th>date</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007-05-01</td>
<td>8:00</td>
</tr>
<tr>
<td>2007-05-02</td>
<td>8:30</td>
</tr>
<tr>
<td>2007-05-03</td>
<td>8:00</td>
</tr>
<tr>
<td>2007-05-04</td>
<td>5:00</td>
</tr>
<tr>
<td>2007-05-07</td>
<td>9:00</td>
</tr>
<tr>
<td>2007-05-08</td>
<td>6:15</td>
</tr>
<tr>
<td>2007-05-09</td>
<td>7:45</td>
</tr>
<tr>
<td>2007-05-10</td>
<td>3:45</td>
</tr>
<tr>
<td>2007-05-11</td>
<td>5:00</td>
</tr>
<tr>
<td>2007-05-12</td>
<td>1:30</td>
</tr>
<tr>
<td>2007-05-14</td>
<td>8:00</td>
</tr>
<tr>
<td>2007-05-15</td>
<td>8:30</td>
</tr>
<tr>
<td>2007-05-16</td>
<td>7:45</td>
</tr>
<tr>
<td>total</td>
<td>87:00</td>
</tr>
</tbody>
</table>

After:
2.1.3 Lessons learned

- In the development of eXtremeManagement I see the same happening as in development of products for clients. There are lots of great ideas at first, but when you actually start working, some take longer to finish than expected, others turn out to be not really possible without major restructuring or going to a newer version of some part of the software stack, old ideas are scrapped in favour of new ideas, priorities change, etcetera. This, in fact, is exactly the sort of environment where Extreme Programming and eXtremeManagement are meant to be used: what seemed like a good idea four months ago may not be the best idea now.

- Adding the Bookings per week overview was just a few hours work. This was possible because the base was right, so doing this correctly pays off. Getting that base right is described in Browser views (section 4.2).

2.2 AJAX

AJAX is a relatively new web technology that combines Javascript and XML to improve the ease and speed of use of a website. In Plone 3.0 a lot is taken care of already. If you have enough rights, you can click on the Title of the eXtremeManagement content types and you can instantly change that title. On Plone 2.5 you need to install extra software (Kinetic Style Sheets or KSS) to do anything. On 3.0 KSS is already available by default.

There were two ideas at the beginning of the assignment.

2.2.1 Adding a Booking

Within a task it is nice to be able to add a booking directly without leaving the page. KSS could be used for that. At the moment this is done with a separate form. Recently though, at Zest we have discovered the gtimelog\(^1\) tool. With this tool you can just keep a log of what you worked on today for how long. Then at the end of the day you just book that in the eXtremeManagement tool in one go with the update_hours_form, instead of going to the individual tasks. I even created a small product willdo\(^2\) inspired by this. For now everyone is happy and the need for making it faster to add a Booking on a Task directly has diminished. Still a nice idea for the future.

---
\(^1\)[http://mg.pov.lt/gtimelog/]
\(^2\)[http://svn.vanrees.org/svn/opensource/willdo/trunk]
CHAPTER 2. USER INTERFACE

2.2.2 Moving Stories

Moving a Story from one iteration to the next should be made simpler. Now you have to go to the first iteration, cut the story, go to the next iteration and paste it. Better would be to display both iterations on one page and let the user drag the story between the two.

This turns out to be very difficult to do in Plone 2.5. Colleague Balasz Ree has some ideas there, as he is a KSS guru. But currently it is not possible. Colleague Mark van Lent has started an eXtremeManagement branch for release plans\(^3\) where he is doing something a bit like this.

Anyway, in light of those remarks it was thought better to skip this part for now. This assignment therefore stays mostly aimed at getting the base of eXtremeManagement right and future proof.

2.2.3 Lessons learned

- These two goals turned out to be too much work to fit in five months together with the other work, as the rest of the work took more time than expected.

- I do not mind some small amount of user interface work, like in Bookings per week (subsection 2.1.2). But I like the ground work more, the coding behind the scenes to get the inner workings of a product right. Plus investigating how to get a product to work with newer technologies like Plone 3.0. I am easily satisfied with how a user interface looks (if it works, it works), but for me getting the core of a product right is the real challenge. The next two chapters describe exactly that.

\(^3\)http://dev.plone.org/collective/browser/eXtremeManagement/branches/markvl-release-plan/
Chapter 3

Plone 3.0

I recently added the biggest part of this chapter on my weblog with the title eXtremeManagement on Plone 3.0\(^1\).

At the beginning of the assignment, eXtremeManagement was fit for Plone versions 2.1 and 2.5. We want it working with Plone 3.0 too, as that is expected to be released in spring or summer 2007. At the time of writing (halfway through June) Plone 3.0 is in beta status. So some things may still change that eXtremeManagement needs to react to. But currently all automatic tests of the eXtremeManagement pass for Plone 2.5 and 3.0. Plus: a test of migrating the current database of the Zest projects site to Plone 3.0 worked fine. So what was done to reach that point?

3.1 General actions

The general steps that every product needs while upgrading from Plone 2.5 to Plone 3.0 are described in the Upgrade guide\(^2\) on plone.org.

Some steps were already taken for Plone 2.5, like setting up the workflows with GenericSetup. Some steps are very simple. An example is that this:

\[
\text{get_transaction().commit(1)}
\]

needs to be replaced with:

\[
\text{transaction.commit(1)}
\]

The other ones that needed checking in eXtremeManagement were:

\(^1\)http://maurits.vanrees.org/weblog/archive/2007/06/extrememanagement-on-plone-3.0
\(^2\)http://plone.org/documentation/manual/upgrade-guide/version/2.5-3.0
• Searching users/groups via the Membership tool is deprecated; it turns out eXtreme-
Management does not use any of the deprecated methods, so fifteen minutes of
checking was all that was needed here.

• Portlets have a new infrastructure. Needed action: check that our classic portlets
still work in Plone 3.0. They turn out to work fine. In the future we may want
to investigate what the benefits of switching to the new portlet system are. A
downside is that this would be incompatible with Plone 2.5, so it would require a
separate branch.

• main_template.pt now uses Zope 3 viewlets. eXtremeManagement is not using
an own version of that template, so no action was needed. In fact, we were not
even using that template, which does not sound like the best idea. So I changed
our templates to use the page layout as given by the main_template.

• The “Sharing” tab is now a global action. In previous Plone versions you had to
explicitly add a Sharing action to your content type if you wanted to see the sharing
tab when viewing that content in the Plone user interface. On the sharing tab you
can share this content with others. Specifically in the case of eXtremeManagement:
we have this tab on Projects, so we can give developers the Employee role and
assign Tasks to them.

On Plone 3.0 this action is now always available. This means that currently some of
our content types actually list this tab twice, as they explicitly add it to themselves
already. The problem now is that you cannot actually do this correctly on Plone
2.5 and 3.0 at the same time: either you have no sharing tab in 2.5 or you have
two sharing tabs in 3.0. Since 3.0 is beta still, I opted to have it working correctly
in Plone 2.5 only. Once Plone 3.0 is officially released, we can easily fix this on
a branch. Or probably: at that point we can move 2.5 compatibility to its own
branch and have trunk specifically for 3.0 support.

3.1.1 Lessons learned

• Keeping an eye on the various plone mailing lists helps: without them I would
not have known that this general upgrade guide existed. Now I even assisted by
adding a comment about an extra step that might be useful to add, pointing to
change set 43188\(^3\). And I know where to look when others ask for help porting
their product to Plone 3.

\(^3\)http://dev.plone.org/collective/changaset/43188
3.2 More fixes

There were some other steps that needed to be taken before eXtremeManagement would work on Plone 3.0. Some might need to be added to the general list and some may be specific for eXtremeManagement.

3.2.1 Viewing content

When you have a Content Management System like Plone it is nice if you can actually see some content. At first try after migrating the current Zest database to Plone 3.0 on a local test instance, none of our content types were actually visible. Change set 43353 lists the changes that were done to make ProjectFolders viewable again. (Change set 43354 has the changes to the other content types.) Some lines could be removed from content/ProjectFolder.py as they were already in the GenericSetup profile. That profile needed some changes too, in this case the file profiles/default/types/ProjectFolder.xml. The most important are the additions of the aliases. Those lines should now look like this:

```xml
<alias from="(Default)" to="base_view"/>
<alias from="edit" to="base_edit"/>
<alias from="properties" to="base_metadata"/>
<alias from="view" to="base_view"/>
```

It could be that this is only needed because our types inherit from OrderedBaseFolder and not from e.g. ATFolder. If your own content type is also not visible on Plone 3.0, I would advise you to add these aliases as a first try.

Actually, the properties alias is superfluous on Plone 3.0 (but not on 2.5, so we will keep it for now). On 3.0 the properties of content can be reached via the normal edit form.

3.2.2 Portlet classes

Some html classes have been deprecated, so you should not use them in your templates anymore. Specifically, get rid of portletItemLast and lastItem in portlets. See change sets 43533 and 43987. And get rid of portletDetails in favour of portletItemDe-

---

4 [http://dev.plone.org/collective/changeset/43353](http://dev.plone.org/collective/changeset/43353)
5 [http://dev.plone.org/collective/changeset/43354](http://dev.plone.org/collective/changeset/43354)
6 [http://dev.plone.org/collective/changeset/43533](http://dev.plone.org/collective/changeset/43533)
7 [http://dev.plone.org/collective/changeset/43987](http://dev.plone.org/collective/changeset/43987)
tails. See change set 43555. Adding a header and footer like in 43535 is also nice:

<dt class="portletHeader">
  <span class="portletTopLeft"></span>
Portlet title
  <span class="portletTopRight"></span>
</dt>

...  
<dd class="portletFooter">
  <span class="portletBottomLeft"></span>
  <span class="portletBottomRight"></span>
</dd>

### 3.2.3 Workflow actions

The general list of things to do when preparing your product for Plone 3.0 already mentions workflows: they need to be done in GenericSetup. That was already done for eXtremeManagement. But this turned out not to be enough. Our content types had a drop down list with workflow actions, but they were not clickable. So you could not do any workflow actions, at least not from the usual drop down workflow menu. This has been put right in change set 43600 by adding a url to the workflow action, instead of having it empty (which worked fine in Plone 2.5). This is the change for profiles/default/workflows/eXtreme_Iteration_Workflow/definition.xml:

```
<transition transition_id="complete" title="Finish"
  new_state="completed">
  <action url="" category="workflow">Finish</action>
  <action url="%(content_url)s/content_status_modify?workflow_action=finish"
    category="workflow">Finish</action>
</transition>
```

But this means that most add-on Products for Plone have no functioning workflow in Plone 3.0, assuming that most Products have an empty url for that action. This also means that when you change the id of a transition id, you also need to remember to change the url, else the old transition is still done or at least tried. In fact, did you

---

8<http://dev.plone.org/collective/changeset/43555>
9<http://dev.plone.org/collective/changeset/43535>
10<http://dev.plone.org/collective/changeset/43600>
notice that exact mistake in the code above? Probably not. I know I did not. :) The transition id is complete, so the action url should have ended with that as well and not with finish.

Can we make this simpler? Yes. I provided a fix for core Plone so it would accept the empty urls that most workflows have. I only just got commit privileges to core Plone, partly based on this proposed fix probably, so change set 15335\textsuperscript{11} is my first actual commit there. This meant that I could revert\textsuperscript{12} my earlier change to eXtremeManagement.

\subsection*{3.2.4 Lessons learned}

- It took a long time before all tests passed. Partly this is because eXtremeManagement has a very decent test coverage. This is a good thing: it isolates the problem to a single failing line in the tests, allowing you to concentrate on one bug at the time. Test driven development like this is good.

- It is very satisfying to finally see all tests pass on Plone 3.0 after all that work.

- Making patches for core Plone and then getting access to commit those changes myself is also very cool.

\footnotesize
\begin{itemize}
\item \textsuperscript{11}http://dev.plone.org/plone/changeset/15335
\item \textsuperscript{12}http://dev.plone.org/collective/changeset/43993
\end{itemize}
Chapter 4

Zope 3

By using new technologies from the next Zope version, the eXtremeManagement tool gets a lot cleaner and will be easier to extend. Declaring and using Interfaces (section 4.1) is the first step. Then we can separate data, presentation and logic cleanly with the help of Browser views (section 4.2). An option was to add a Utility (section 4.3) for getting all employees that work on a project; this was abandoned as it was not a really useful idea after all. With the eXtremeManagement landscape cleaned up in the previous sections, we can now do a big overhaul under the surface with Annotations for time registration (section 4.4), which was not a part of the original assignment, but very much needed anyway.

4.1 Interfaces

In Zope 3, interfaces are important. Interfaces are classes that define functionality. Other classes that claim to implement that interface should provide that functionality.

4.1.1 Marker interfaces

The simplest kind of interface is the marker interface. An example is in interfaces/xm-story.py:

```python
from zope.interface import Interface
class IXMStory(Interface):
    """eXtremeManagement Story
    ""
```
It is called a marker interface because you can mark a class with it. Marking means that you claim that a certain class implements this interface. We do that in content/Story.py:

```python
from Products.eXtremeManagement.interfaces import IXMStory
...
class Story(OrderedBaseFolder):
...
    implements(IXMStory)
```

On its own this does absolutely nothing. But now we can do something with this interface in other parts of our code. For example, in browser/configure.zcml we register a certain view or page only for objects that implement this IXMStory interface:

```xml
<page
    name="story"
    class="stories.StoryView"
    for="Products.eXtremeManagement.interfaces.IXMStory"
    permission="zope2.View"
    allowed_interface=".interfaces.IStoryView"
/>
```

Details aside, this now means that the page with the name story is available exclusively for objects implementing that interface, instead of all objects, which would be much less clean.

As a first step it is enough to define marker interfaces for all our content types. So in the interfaces/ directory we have defined interfaces IXMProject, IXMIteration, etcetera.

### 4.1.2 Functional interfaces

Interfaces can be much more than just markers, though. They can define functionality. So let’s call them functional interfaces, although that is not an official term. Objects that implement this interface promise to provide that functionality. In the file timing/interfaces.py we define an interface like this:

```python
from zope.interface import Interface
from zope.interface import Attribute

class IActualHours(Interface):
```
"Actual hours and minutes worked"

actual_time = Attribute("Actual time")

def recalc():
    """Recalculate the total of bookings/actual hours."
    """

Any class that claims to implement this IActualHours interface must at least have an attribute actual_time and a method recalc. This is true for our Booking class, so we can claim to implement the interface. Instead of saying this in python code, like in the example for IXMStory above, let's now say this in timing/configure.zcml:

<class class="Products.eXtremeManagement.content.Booking">
    <implements interface=".interfaces.IActualHours" />
</class>

We will later see how this is used in practice.

4.1.3 Lessons learned

- Adding interfaces is pretty easy.

- I should add more info to those basic interfaces to make them even more useful.
  This is already done for the interfaces in browser/interfaces.by.

4.2 Browser views

When creating a Product like eXtremeManagement it is a good idea to separate data, presentation and logic. The obvious way to do that is by using the Model-View-Controller approach. That is a fancy way of saying the same thing: data, presentation and logic. In Zope and Plone content types are the Models: they contain the data and should ideally be as simple as possible. In eXtremeManagement examples are Project and Iteration. The Views are the html pages or in Zope terms: page templates. The controllers are various python scripts that take care of the logic.
4.2.1 Model

The content types of eXtremeManagement are specified in python files in the `content/` directory. The most important function of those files is to define the database schema: which fields does a Project have? How should they be stored in the Zope database? Do they have default values? So far they are not very different from table definitions in a relational database. But Zope uses an object database. Zope does not store a row or record in a table, but it stores an object. This works very well and is one of the strengths of Zope. It should be used with care though. You could be tempted to add a lot of logic to objects in the form of methods. Again: this is powerful and useful, but as one of the principles of python says: Simple is better than complex. Often Controllers are a better place to put logic.

A simple example: a Story has a method `get_progress_perc`. This calculates the progress percentage. Uninteresting details aside: if the Story was estimated at four days and the implementation has so far taken 1 day then the progress percentage is 25. Great: you can ask a Story what its progress percentage is!

But why should that logic be put in the Story class? In this case it makes more sense to put this as a separate function in a python utility module. This keeps the logic out of the content classes. And it avoids duplication, as this function can now be used for other uses as well. For instance: a Task also has an estimate and a field for total hours worked, so this progress percentage could get used there as well.

Now a Controller (a browser view, which we will learn more about below) can ask a Story (or Task or something else) its estimate and its current total hours worked and give those numbers to the function in the utility module. This puts the logic where it belongs: in the Controller. An added bonus is that the new function is much easier to test: it is basic mathematics with two numbers instead of with a Story.

Now, a few improvements like that can be made. But actually the biggest improvement in making the eXtremeManagement content types simpler, is by introducing annotations and events, which will be explained below. What I will say now already, is that this meant that our content types could get a lot simpler. For example, when a Booking (hours worked) gets added to a Task, the Task should update its actual hours worked: the total of all its Bookings.

Or actually: the Task should make sure this total value is updated in the Zope Catalog: that is a central place where the most important info about all objects is kept. It is much faster to access this Catalog than the objects themselves, so it is a good idea to use the Catalog whenever possible. It should be kept up to date though when the objects change. To do this, at first the Booking class needed two methods `manage_afterAdd`
and \texttt{reindexObject}. After calling in the help of annotations and events, these methods are no longer necessary.

In fact, when I started this final assignment, the content types had fifty methods. Now they have twenty five. Everything still works, so I would say that is a tangible improvement: simple is better than complex.

\subsection{View}

We have seen what the Models are: the content types like Project, Iteration, etcetera. Now it is time for the Views. They are simply the html pages that are presented to the user by the Zope server.

In Zope the html pages are represented by page templates. They contain plain html, but with some extra tags that can be used to inject data from the Model or the Controller. A simple example: the \texttt{story\_view.pt} file has this code that inserts the description of the story into the html page:

\begin{verbatim}
<div tal:condition="here/Description"
    tal:content="here/Description">
  description
</div>
\end{verbatim}

This code means: if \texttt{(tal:condition)} the object that is currently viewed (\texttt{here}) has a Description, then put that Description as content \texttt{(tal:content)} inside the \texttt{div}.

So Zope page templates can have some logic. In simple cases like this that is fine. But you should not overdo it. Small python snippets can also be used in page templates. That can be very handy and sometimes it is no problem at all and it would be overkill to do this anywhere else. But page templates are meant to be understandable for web developers who only know html, css and maybe Javascript. Having difficult python code inside such a template will scare them into not touching that file at all. Or if it does not scare them, they will be tempted to change that python line as well and horribly mess it up. Having code like this inside a page template (here the \texttt{booking\_year.pt}) is wrong:

\begin{verbatim}
<tal:month tal:repeat="dmonth python: range(0,12)">
  <tal:block tal:define="
    global total\_monthly python: 0;
    month python: base\_month-dmonth;
    year python: test(month<1, base\_year-1, base\_year);
    month python: test(month<1, month+12, month);">
    That was before, this is after:
\end{verbatim}
CHAPTER 4. ZOPE 3

That is a lot cleaner. Before, that template was 82 lines long. After, it has just 39 lines. Again: simple is better than complex.

Another statistic: before, the word python was mentioned 156 times in our page templates. After, it is mentioned just 68 times. A marked improvement, as this makes the templates easier to understand for non-programmers.

4.2.3 Controller

After Model and View we now look at eXtremeManagement from the viewpoint of the Controller. I already mentioned that the Models and Views have gotten simpler. That almost automatically means that the controllers have gotten more complex. But as the python philosophy says: complex is better than complicated. Logic has moved from Model and View to Controller and that is exactly where the logic should be. Controllers exist for the sole reason of providing logic.

But where are those Controllers? What are they in practice? Well, most of the logic used to be in python scripts inside the skins/ directory. These are special scripts that have one advantage over normal python files: they are customisable through the web: if you are logged in as a user with the right permissions, you can view such a script, change it and it will immediately have effect on the website. They also have downsides: they are slow, they are hard to test and it is hard to inspect them with the python debugger.

That is reason enough to want to get rid of them. And that is just what I did. At the beginning there were 23 python scripts in the skins/ directory. Now there are only five. The rest has been converted into faster, testable and debuggable browser views.

Browser views are brought to you by new Zope 3 technology and is usable in Zope 2 today. Browser views are classes. They are usually defined in browser.py or in a python file inside the browser/ directory. In browser/xmbase.py a browser view is defined that is used as a base class for several other browser views:

```python
from Products.CMFCore.utils import getToolByName
from Products.Five.browser import BrowserView
from Acquisition import aq_inner
```
class XMBaseView(BrowserView):
    """Base view for showing info about an object."
    """

def __init__(self, context, request):
    super(XMBaseView, self).__init__(context, request)
    context = aq_inner(context)
    self.xt = getToolByName(context, 'xm_tool')

def main(self):
    """Get a dict with info from this object."
    """
    return {}  

Not every line is interesting to discuss. But what should be clear from the signature of the __init__ method, is that an instance of the browser view is created (the class is instantiated) by passing it a context (e.g. a Project or an Iteration) and an HTTP request from a browser. That makes them adapters: they adapt a context and a request into a new object.

In an earlier code example, we saw that the booking_year.pt template now defines view as being context/@yearbookings. The context in that case is the Plone website. @@yearbookings points to a browser view that is defined in browser/configure.zcml:

<page
    name="yearbookings"
    class=".bookings.YearBookingOverview"
    for="*"
    permission="zope2.View"
    allowed_interface=".interfaces.IYearBookingOverview"
/>

This xml snippet means that the yearbookings browser view is defined in class YearBookingOverview in the bookings.py file in the current directory, it is defined for all contexts, everyone can view it, and the attributes and methods that can be viewed are defined in the IYearBookingOverview interface in the interfaces.py file in the current directory.

Look again at the part of the booking_year.pt page template that was printed above:
This is the part of the template (View) that talks to the browser view (Controller). A request is sent to get the `yearbookings` view for the current context. This is put in the variable `view`. Then some questions are asked of that view and the answers are stored in other variables, that are used further on in the page template. This is an improvement when compared to the earlier situation where several computations and tests in python were done in the page template.

Like I said, eXtremeManagement has in a few months moved from 23 to 5 python scripts. Two are scripts that are only used on a page template that is not actually in use now but that I do not want to delete just yet. The other three are special controller and validator python scripts that are less easy to turn into browser views, if that is possible at all. They work fine currently. All in all, this means that the current code is more testable and has a cleaner separation between presentation and logic.

### 4.2.4 Lessons learned

- Replacing untestable python scripts in the `skins` directory by testable browser views does wonders for making a product more robust. I can now more easily try something out and run the tests to see if I broke anything. More tests are still welcome of course.

- Really sitting down and doing this takes a lot of time and patience. This is easy to underestimate, which is exactly what I did. It is worth it though.

### 4.3 Utility

An eXtremeManagement Project has a method `getMembers`. You can use that to get a list of members with a specific role in this Project, by default: Employee. At the beginning of this final assignment I had an idea to add a Zope 3 style utility that would be used for this instead. This would mean that other programmers could provide their own version of this utility: they would get an easy hook to influence eXtremeManagement from the code in their own product.
But Projects already have a field `includeGlobalMembers`. By default this is set to True: if you have a certain role in the entire site, then you get listed with the members of this Project, next to persons with that role only on this Project. If this is set to False, then only persons that are explicitly given that role on this Project get added to the list. I have had no complaints about this, so apparently this works to the satisfaction of everybody. It is therefore not necessary to add a special utility for this: simple is better than complex.

4.3.1 Lessons learned

- Do not add complexity for the single reason of trying out a new technology.

4.4 Annotations for time registration

This section was first published in an email to the eXtremeManagement mailing list and on my weblog\(^1\).

eXtremeManagement has gotten an overhaul under the surface. To keep track of actual hours booked and a total of estimates for an Iteration or Story or Task we were using ComputedFields and some ugly methods. We got rid of those. Instead we are now using clean zope 3 style annotations. Why, you ask?

4.4.1 Reasons

- This way we can get rid of several kludgey methods on our content types, like `manage_afterAdd` and `reindexObject`. Using events, like already started on trunk, also helps here.

- We get rid of ComputedFields like `getRawActualHours`. On a Booking this was okay: it just changed 1 hours plus 30 minutes into 1.5 hours; this is still done actually, but with a small twist. But on a Task this would calculate the total of `getRawActualHours` of all its children, either by waking up these objects or by querying the catalog. Instead, with annotations and events we now make sure that this total is only calculated when a Booking has been added or changed or moved.

- During a reinstall GenericSetup sees that we have some indexes in our catalog.xml so it unhelpfully empties those indexes. So we need to reindex them during our

\(^1\)http://maurits.vanrees.org/weblog/archive/2007/06/annotations-for-time-registration
reinstall. This always seemed to have a negative effect on the consistency of booked hours and estimates in the catalog. At least it seemed to help to manually recatalog first the Bookings, then the Tasks and then the Stories and possibly the Iterations after a reinstall.

GenericSetup only empties the indexes though and not the metadata columns, so this should not have been necessary. Maybe I was misled through coincidences into believing that this helped. Anyway, now everything is stored on the object itself or in annotations, so the order in which you recatalog the content types really does not matter anymore. :)

- Our content types are simpler now.
  - A Task does not need to be aware of any changes to its Bookings and it does not need to inform its parent Story that its estimate has changed. Keeping everything updated is now the responsibility of the zope 3 events.
  - Information about children (total hours booked or estimated) is now stored separately in annotations.

In general, this approach makes me feel sane again. Plus it allows me to play with annotations and they are cool. ;-)

4.4.2 Implementation

- The annotations are taken care of in the timing/ directory.

- A few interfaces have been introduced there:
  - IActualHours: this is for content types that wish to store actual hours. Bookings actually implement this functionality already, so when you adapt a Booking to the IActualHours interface the Booking itself is returned.
  - IActualHoursContainer: This is a marker interface that says: this object has children with actual hours. Iterations, Stories and (Poi)Tasks are marked with this. Actually, marker interface may be a bad choice of words as it does have one method that needs to be provided: contentValues. Since the types mentioned are folderish (folder-like) types they provide this method already. Objects providing this interface should also be adaptable to IActualHours.
  - IEstimate: this is for content types that wish to store estimates. (Poi)Tasks actually implement this functionality already, so when you adapt a Task to the IEstimate interface the Task itself is returned.
– **IEstimateContainer**: This is a marker interface that says: this object has children with estimates. Iterations and Stories are marked with this. The same remarks as with **IActualHoursContainer** can be made. Objects providing this interface should also be adaptable to **IEstimate**.

- By adapting an object to **IActualHours** or **IEstimate** you get an object that has a **recalc** method that recalculates its actual hours or the estimate and stores that in an attribute. For the Containers this is stored in annotations.

- For Booking this is stored in a property that functions as a computed field. The **recalc** method then only needs to reindex the object:

```python
@property
def actual_time(self):
    return self.getHours() + (self.getMinutes() / 60.0)

security.declarePublic('recalc')
def recalc(self):
    self.reindexObject(idxs=['actual_time'])
```

- The same is done for the estimate of Tasks.

- For the various Containers the **recalc** method fetches its children. It does not matter if they are Bookings, Tasks or Stories. It then adapts them to the **IActualHours** or **IEstimate** interface and adds the value of the stored property to its own total. For **IActualHoursContainers** it looks like this:

```python
def recalc(self):
    """Recalculate the total booked hours for this container."
    """
    context = self.context
    total = 0.0
    for obj in context.contentValues():
        actual = IActualHours(obj, None)
        if actual is not None:
            total += actual.actual_time
    self.actual_time = total
    context.reindexObject(idxs=['actual_time'])
```
• Zope 3 events are used to arrange that everything is updated at the right time, namely when a relevant object is added, moved, removed or changed. Event handlers make sure that the value for the object itself is recalculated and that its parents, grandparent, etcetera are also recalculated. This uses the following code, where adapter is either IActualHours or IEstimate:

```python
def recursivelyRecalc(object, adapter):
    current = aq_inner(object)
    anno = adapter(current, None)
    while anno is not None:
        anno.recalc()
        current = aq_parent(current)
        anno = adapter(current, None)
```

• This value is also put in the catalog:

```python
from Products.CMFPlone import CatalogTool as catalogtool

def actual(object, portal, **kw):
    anno = IActualHours(object, None)
    if anno is not None:
        return anno.actual_time
    return None

catalogtool.registerIndexableAttribute('actual_time', actual)
```

### 4.4.3 Extra cleanup

While I was busy, I did some related cleanup:

• Indexes and metadata for the catalog are no longer registered in the code for the content types themselves (so the content/ directory). This is now in the Generic-Setup profile in catalog.xml. The former code might not even work anymore in newer versions of Plone or Archetypes, if I am informed correctly.

• In the course of time we registered far too many indexes and metadata for our content types. Most indexes were unneeded and have been removed, except getAs-
signees and getBookingDate. By reinstalling or applying the GenericSetup profile the others will be removed if you use Plone 2.5.3 (GenericSetup 1.2). Removing metadata columns from the catalog is not automatically possibly currently with GenericSetup. I submitted a patch\(^1\) for that. But the following are unused and you can manually remove them: getRawActualHours, getRawDifference, getRawEstimate. Two of those have simply gotten a different name though: actual_time and estimate. The new names feel more pythonic. :) It seemed logical to change the names while I was at it.

- When you migrate the schema of Tasks then under the hood all your Tasks are created fresh. This means that lots of emails will be sent out to inform people that some tasks have been assigned to them. This is bad and I apologise for the times in the past when this went wrong while I was testing and I ended up spamming my colleagues... :) Our reinstall actually knows this and works around it by setting a boolean value that tells Tasks to not send those emails. This functionality has been there for a while, but you can now also flip this switch manually: in the portal_properties tool there is a property sheet xm_properties with a setting send_task_mails that controls this.

### 4.4.4 Lessons learned

- I started the switch to annotations on a subversion branch, which is standard practice with such a heavy change. But then I started doing other changes on that branch that were needed (or at least useful) for the annotations, like restructuring and cleaning up the tests and trying a new idea for migration of content. That took a long time. It was really needed, but it would have been better to do those activities on a separate branch first, then merge that to trunk, and then continue with the work on annotations. I ended up doing something like that anyway, but with more headaches than needed. So: if you are doing separate refactorings, do them on separate branches.

- When submitting a patch to an upstream product like GenericSetup, always provide a test that proves your fix has the wanted effect. That helps a lot in getting your patch accepted.

---

\(^1\) That was recently accepted into GenericSetup trunk. See issue 483 in the Zope collector.
Chapter 5

Conclusion

eXtremeManagement has been given a thorough cleanup. It runs on the current Plone 3.0 beta. Where useful, Zope 3 technologies have been put to good use. Personally, I have learned a lot about these new software versions and I am sharing that knowledge on my weblog and in mailing lists. At Zest Software we are happily using the latest version of eXtremeManagement and are full of ideas for further improvement.

Some recommendations and further actions to take:

- Release a new version of eXtremeManagement soon: it is ready.

- Building on the foundation laid during this assignment, do more work on the user interface. There are lots of ideas floating around. See Future plans (Appendix A).

- Once Plone 3.0 is officially out, copy the current subversion trunk to a maintenance branch and continue development for Plone 3.0 only on trunk.

In closing, I will say I had a great time with eXtremeManagement and its users and co-developers on the mailing list. eXtremeManagement is a rocking product, ready to handle the future.
Samenvatting

(Summary in Dutch)

In dit rapport doe ik verslag van mijn afstudeeropdracht voor de studie Informatica aan de Hogeschool Rotterdam. Ik verbeter de eXtremeManagement tool: een project management tool gebaseerd op Extreme Programming principes en draaiend op het content management systeem Plone.

De Introductie (hoofdstuk 1) schetst de achtergrond van deze opdracht. Wat is Zope? Wat is Plone? Wat is eXtremeManagement? Hoe passen ze in elkaar?

Ik heb wat verbeteringen aangebracht in de gebruikersinterface (hoofdstuk 2). De meeste oorspronkelijke plannen werden niet uitgevoerd, om redenen uiteenlopend van simpel tijdgebrek tot nieuwe inzichten die het originele idee onderuit haalden. De focus van de opdracht verschoof daarmee naar het verbeteren van de kern, in plaats van het mooi maken van de buitenkant.

Plone 3.0 (hoofdstuk 3) vertelt het verhaal van het geschikt maken van eXtreme-Management voor de nieuwe versie 3.0 van Plone, die overigens nog de beta status heeft. Ik heb wat standaardaanpassingen doorgevoerd die alle Plone producten van derden moeten doen. Ik heb ook andere veranderingen gedaan die nodig bleken te zijn. Tenslotte heb ik een verbetering aan de kern van Plone bijgedragen om deze aanpassingen voor 3.0 simpeler te maken voor andere producten.

Met Zope 3 (hoofdstuk 4) komen we bij het hart van de zaak. Meer dan oorspronkelijk gedacht, moest de focus op dit deel gericht worden. Ik heb markeer- en functionele interfaces toegevoegd. Ik heb browser views gemaakt om een duidelijker scheiding te brengen tussen data, presentatie en logica (Model-View-Controller). Ik heb annotaties ingebracht voor het bijhouden van inschattingen en gewerkte uren. Deze wijzigingen leveren samen een veel schonere versie van eXtremeManagement op dan er was bij de start van de afstudeeropdracht.

Ik trek de conclusie (hoofdstuk 5) dat eXtremeManagement schoon en toekomst klaar is en dat ik veel heb geleerd bij deze opdracht.
Appendix A

Future plans

As a small sneak preview of what the future might hold for eXtremeManagement, the next two pages have screen shots of the Zest projects site. They show the main parts of two Iterations, displaying Stories that colleagues have partially already started to work on.
<table>
<thead>
<tr>
<th>Story</th>
<th>Status</th>
<th>Rough Estimate</th>
<th>Estimate</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>[↓] Release Plan: Add stories in project view</td>
<td>Completed</td>
<td>1.5</td>
<td>12.00</td>
<td>27.00</td>
</tr>
<tr>
<td>[↓] Release Plan: Estimating and Prioritizing stories</td>
<td>Estimated</td>
<td>4.0</td>
<td>32.00</td>
<td>8.15</td>
</tr>
<tr>
<td>[↓] Customers and employees can create an initial Release Plan (startup sprint)</td>
<td>In progress</td>
<td>5.0</td>
<td>26.00</td>
<td>8.15</td>
</tr>
<tr>
<td>[↓] Customer can manage priority of stories in the Release Plan (subsequent iterations)</td>
<td>Estimated</td>
<td>2.5</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>[↓] A developer should be able to create tasks for a set of bug tracker issues within a story.</td>
<td>Estimated</td>
<td>2.0</td>
<td>11.00</td>
<td>8.00</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>15.0</td>
<td>81.00</td>
<td>51.30</td>
</tr>
<tr>
<td>story</td>
<td>status</td>
<td>rough estimate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>-----------</td>
<td>----------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ ↓ ] Project leader can plan iterations</td>
<td>Estimated</td>
<td>2.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ ↓ ] Customer, employees and project leader can view progress reports at iteration level.</td>
<td>in progress</td>
<td>3.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ ↓ ] Customer, employees and project leader can view progress reports at Project level</td>
<td>Estimated</td>
<td>3.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ ↓ ] Track remaining hour more accurately</td>
<td>Estimated</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ ↓ ] Customer and employee can monitor estimated and actual project velocity</td>
<td>Estimated</td>
<td>0.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ ↓ ] User Interface</td>
<td>Estimated</td>
<td>10.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ ↓ ] A developer can monitor his personal velocity</td>
<td>Estimated</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ ↓ ] Stories should have other fields</td>
<td>Estimated</td>
<td>1.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ ↓ ] Time registration form improvements</td>
<td>Estimated</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ ↓ ] investigate integration of gtimelog</td>
<td>Estimated</td>
<td>2.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ ↓ ] When defining tasks for a story, use a collapsible area to add a task</td>
<td>Estimated</td>
<td>0.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ ↓ ] As a customer I want to be able to add screenshot or other files to a story.</td>
<td>Estimated</td>
<td>0.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ ↓ ] As a manager I want to be able to label a project or iteration as unbillable</td>
<td>Estimated</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ ↓ ] As a developer I want to be able to select a tasks from a list of 'Unassigned tasks' and assign it to myself</td>
<td>Estimated</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>total</td>
<td></td>
<td>28.5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>