Plat_Forms 2012 Task: CafMan

Ulrich Stärk, ulrich.staerk@fu-berlin.de
Lutz Prechelt, prechelt@inf.fu-berlin.de

Institut für Informatik
Freie Universität Berlin
Berlin, Germany
http://www.inf.fu-berlin.de/w/SE/

October 2012

Abstract

This document contains the requirements for the system to be built by the participants of the Plat_Forms 2012 contest. The system is called CafMan - The Caffeine Manager. It is to be implemented within two days by a team of three people. For further details about the contest, please see www.plat-forms.org.

Contents

1 Introduction 4

1.1 Theme/purpose of CafMan . . . 4

1.2 Requirements priorities and notation . . . 4

2 Functional requirements: General 5

3 Functional requirements: Use Cases 6

3.1 User actions overview . . . 6

3.1.1 Main scenario . . . 6

3.1.2 Exceptions and variants . . . 6

3.1.3 Notes . . . 7

3.2 User signup . . . 7

3.2.1 Main scenario . . . 7

3.2.2 Exceptions and variants . . . 7

3.2.3 Notes . . . 7

3.3 Join a coffee kitty . . . 8

3.3.1 Main scenario . . . 8

3.3.2 Exception and variants . . . 8

3.3.3 Notes . . . 8

3.4 Create a coffee kitty . . . 8

3.4.1 Main scenario . . . 9

3.4.2 Notes . . . 9

3.5 Coffee consumption . . . 9

3.5.1 Main scenario . . . 9

3.5.2 Exceptions and variants . . . 9

3.5.3 Notes . . . 10

3.6 Monitoring Caffeine Levels . . . 10

3.6.1 Main scenario . . . 10

3.6.2 Exceptions and variants . . . 10

3.6.3 Notes . . . 10

3.7 Consulting The Highscores . . . 11

3.7.1 Main scenario . . . 11

3.7.2 Exceptions and variants . . . 12

3.8 Managing Coffee Kitties . . . 12

3.8.1 Main scenario . . . 12

3.8.2 Exceptions and variants . . . 12

3.9 Administering a Coffee Kitty . . . 12

3.9.1 Main scenario . . . 13

3.9.2 Notes . . . 13

3.10 User account management . . . 13
3.10.1 Main scenario . . . 13
3.10.2 Notes . . . 14
3.11 Resetting the Application . . . 14
3.11.1 Main scenario . . . 14

4 Non-functional requirements **15**
4.1 Quality . . . 15
4.2 Scalability . . . 15
4.3 Programming style and documentation . . . 15
4.4 Persistence and State . . . 16

5 Rules for development **17**
5.1 What is allowed . . . 17
5.2 What is not allowed . . . 17
5.3 Intermediate versions and user feedback . . . 17
5.3.1 How to post a message in the blog . . . 18
5.4 Talking to the customer . . . 18
5.5 Delivery . . . 19

1 Introduction

1.1 Theme/purpose of CafMan

CafMan - The Caffeine Manager allows its users to manage and monitor their caffeine intoxication and helps them to administer their team's coffee kitty. Users participate in coffee kitties, consume coffee, monitor their caffeine levels, and set up notifications. Kitty administrators create coffee kitties, register payments, and set prices.

1.2 Requirements priorities and notation

The requirements described in this document are categorized into three different priority levels and each requirement is marked accordingly:

MUST	MUST marks an essential requirement. Unless all of these requirements are implemented, the system is considered inacceptable.
SHOULD	SHOULD marks an important requirement. If some of these requirements are not implemented, the system is considered incomplete, but acceptable.
MAY	MAY marks an optional requirement. These requirements are considered nice-to-have but need not be implemented when time is short or if their cost-benefit ratio is considered too high. Always try to maximize the number of MUST requirements implemented before working on MAY requirements. Especially always try to improve quality (see 4.1) before tackling complicated MAY requirements.

Each requirement is marked by exactly one of these three terms, followed by a subscript number that is the unique reference number of that requirement (also repeated in the margin).

2 Functional requirements: General

You will be provided with static HTML prototypes of CafMan that are based on Twitter Bootstrap. You are not required to use them but encouraged to do so in order to have a somewhat comparable look and feel in all solutions. However, you MUST$_1$ use the same menu and page structure as in the static prototype. The names of the pages that a given use case is supposed to appear on is given in the use case description. This name also serves as the name for the menu link. It is not required to use it in the URL to the page. M 1

Wherever in the prototype ID attributes are present for HTML elements, you MUST$_2$ also use them in your implementation. This will mostly be the case for hyperlinks, form fields, and buttons. M 2

In cases where links, form fields, and buttons are dynamically generated and may occur multiple times, you MUST$_3$ use special CSS classes to identify these elements: M 3

member-request For the element that triggers a request for membership from the coffee kitty search results (see 3.3).

statistic-link For the links to users' caffeine statistics pages from the highscores (see 3.7).

administer-link For the links to the coffee kitty administration pages (see 3.8).

accept-member For the element inside a notification that accepts a request for membership (see 3.9).

decline-member For the element inside a notification that accepts a request for membership (see 3.9).

input-payment For the elements used to record users' payments on the coffee kitty administration page (see 3.9).

Lastly, you also MUST$_4$ name all links and buttons exactly as in the prototype. M 4

Your solution MUST$_5$ be available on port 80 of your server machine and MUST$_6$ be available under the root URL for that machine, e.g. http://d-2012.plat-forms.org/ or http://10.0.0.2/. Also notice that the quality of your deployment will be evaluated as well (see 4.1), so make sure that it is production-ready. M 5 M 6

3 Functional requirements: Use Cases

Each use case consists of three subsections and an introduction: The introduction lists any existing preconditions, gives the names of one or more files from the HTML prototype that correspond to that use case, and states on what page, i.e. under what menu entry, the use case must be available. The mandatory main scenario section describes the standard sequence of events for the use case. The optional exceptions and variants section describes what happens in important error cases and which optional deviations from the main scenario should be considered. The optional notes section provides detail information where needed.

3.1 User actions overview

3.1.1 Main scenario

1. The *user signs up* (3.2).
2. The *user joins a coffee kitty* (3.3).
3. The *user helps themselves to a coffee fix* (3.5).
4. The *user examines their caffeine levels* (3.6).
5. The *user consults the highscores* (3.7).
6. The *user manages their coffee kitties* (3.8).
7. The *user administers the coffee kitty* (3.9).
8. The member logs out (MUST$_7$). M 7

3.1.2 Exceptions and variants

- 1b. The member logs in with their email address and their password (available on the *"Home"* page, MUST$_8$). M 8

3.1.3 Notes

After a successful login, the *"Home"* page changes to display the contents of the *Join a coffee kitty* (see 3.3) or *Coffee consumption* (see 3.5) use cases, depending on whether the user is already a member of a coffee kitty or not.

3.2 User signup

Precondition: The user is not logged in.

Reference: *index.html*

Available on page: *"Home"* (MUST$_9$) M 9

3.2.1 Main scenario

1. The user enters the following mandatory information:
   — full name, display name (MUST$_{10}$) M 10
   — email address, password (MUST$_{11}$) M 11
   — whether to show the display name in highscores and make caffeine statistics publicly available (MUST$_{12}$) M 12

2. The user submits this data for registration.

3. The portal validates the data, registers the user as a new member, stores the data, and logs the user in (MUST$_{13}$). M 13

3.2.2 Exceptions and variants

- 3b. The portal rejects the display name (MUST$_{14}$) or the email address (MUST$_{15}$) because it is not unique (already in use) and sends the user back to step 1. This MAY$_{16}$ be done without the need for a full page refresh, using Ajax. M 14 M 15 m 16

3.2.3 Notes

After a successful registration, the *"Home"* page changes to show the contents of the *Join a coffee kitty* use case (see 3.3).

The portal SHOULD$_{17}$ validate the email address to be syntactically correct. It MAY$_{18}$ validate the email address to have a valid top level domain. It MAY$_{19}$ also validate the email address to have a valid second level domain. Both MAY$_{20}$ be done using Ajax. S 17 m 18 m 19 m 20

The user's caffeine statistics (see 3.6) are hidden from other users of the software and instead of their display names a placeholder is shown in the highscores (see 3.7) unless the user agrees to make this data public. If they choose so, their statistics are publicly available under their display name and their display name also shows in the highscores. Their full name is never shown, except to the kitty administrator (see 3.9).

3.3 Join a coffee kitty

Precondition: The user is logged in.

Reference: *index-authd-no-kitty.html, kitty.html*

M 21 M 22 Available on pages: *"Home"* (when not a member of any coffee kitty yet, MUST$_{21}$) and *"Coffee Kitty"* (MUST$_{22}$)

3.3.1 Main scenario

M 23 1. The user enters a search term into a search form and submits the form (MUST$_{23}$).

M 24 2. The portal searches for coffee kitties whose names match the search term and displays a list of matching coffee kitties (MUST$_{24}$)

M 25 3. The user requests to become a member of a coffee kitty from the results list (MUST$_{25}$).

3.3.2 Exception and variants

m 26 • 3b. This MAY$_{26}$ be done without the need for a full page refresh, using AJAX.

3.3.3 Notes

The first coffee kitty that a user successfully joins or creates, automatically becomes their default kitty (see 3.10).

3.4 Create a coffee kitty

Precondition: The user is logged in.

Reference: *index-authd-no-kitty.html, kitty.html*

M 27 M 28 Available on pages: *"Home"* (when not a member of any coffee kitty yet MUST$_{27}$) and *"Coffee Kitty"* (MUST$_{28}$)

3.4.1 Main scenario

1. The user enters the name of a new coffee kitty into a form and submits it (MUST_{29}). M 29
2. The application validates the name (MUST_{30} be unique), creates the kitty, and makes the user the administrator for that kitty (MUST_{31}). M 30 M 31
3. The portal redirects the user to the administrative page for the newly created kitty (see 3.9) (MUST_{32}). M 32

3.4.2 Notes

The first coffee kitty that a user successfully joins or creates, automatically becomes their default kitty (see 3.10).

3.5 Coffee consumption

Precondition: The user is logged in and member of at least one coffee kitty.

Reference: *index-authd.html*

Available on page: *"Home"* (MUST_{33}) M 33

3.5.1 Main scenario

1. The user chooses the coffee kitty which they are going to use (MUST_{34}). M 34
2. The portal tells the user how much the cup of coffee they are about to consume will cost them (MUST_{35}). M 35
3. The user tells the software that they consumed a cup of coffee (MUST_{36}). M 36
4. The portal updates the user's account balance (MUST_{37}) and their caffeine level (MUST_{38}) and returns them to the *"Home"* page (MUST_{39}). M 37 M 38 M 39

3.5.2 Exceptions and variants

- 1b. Instead of choosing a coffee kitty, the user's default coffee kitty that can be set on the user's profile page (see 3.10) is used (MUST_{40}). M 40
- 1c. Instead of requiring a full page reload when switching the active coffee kitty, the application MAY_{41} use AJAX. m 41

3.5.3 Notes

M 42 This functionality is only available to users that are a member of at least one coffee kitty (MUST$_{42}$).

3.6 Monitoring Caffeine Levels

Reference: *caffeine.html*

M 43 Available on page: *"Caffeine Statistics"* (MUST$_{43}$)

3.6.1 Main scenario

1. The user navigates to the *"Caffeine Statistics"* page of a specific user, either from the menu (own statistics) or from the highscores (somebody else's statistics, see 3.7).

2. The application shows the display name of the chosen user and lets the user download the
M 44 chosen user's caffeine levels during the last 24 hours (MUST$_{44}$) and the last 7 days (MUST$_{45}$)
M 45 as CSV files (see Notes), each on a 30-minute interval (MUST$_{46}$), starting from the current time
M 46 (MUST$_{47}$).
M 47

3. S 48 The application displays the data from the CSV files graphically (SHOULD$_{48}$). For smoother
m 49 graphs, the application MAY$_{49}$ use a higher resolution than the 30-minute interval.

4. S 50 The loggedin user enters a value (in milligrams) into a form field and submits the form (SHOULD$_{50}$).
The portal validates the data, stores it, and sends the user an email alert to his email address the
S 51 moment the user's caffeine level drops below the specified threshold (SHOULD$_{51}$).

3.6.2 Exceptions and variants

- 4b. If the user consults the caffeine levels of a user other than themselves, the application
M 52 MUST$_{52}$ not show the form to set up an email alert.

3.6.3 Notes

This use case is available both for loggedin and anonymous users of the software. Anonymous users can access it from the highscores, loggedin users from the highscores or via a link to their own caffeine statistics in the main menu.

M 53 The download format MUST$_{53}$ be comma separated values (CSV). The file MUST$_{54}$ contain a header
M 54 row for the two columns "timestamp" and "level". The timestamps MUST$_{55}$ be quoted using double
M 55 quotes ("), and the delimiter MUST$_{56}$ be the comma character (,). Sample files are given in *daily.csv*
M 56

and *weekly.csv*.

The caffeine content of coffee products varies wildly depending on the raw product, the preparation method, and other factors. For example, a typical cup of drip coffee (ca. 200 ml) contains between 115 and 175 mg of caffeine while the same amount of percolated coffee only contains between 80 and 135 mg. For simplicity we assume that one unit of coffee always contains 150 mg of caffeine (MUST$_{57}$). M 57

We also assume that the peak caffeine level is reached one hour after ingestion and that the level increases linear. We further assume that the half-time of caffeine is exactly five hours. This means that the amount of caffeine in the human body is reduced by half every five hours.

Your applications MUST$_{58}$ use the following two formulae to calculate the current caffeine level: M 58

$$N(t) = N_0 * t * 0.5^{\frac{t}{5}}, if\ t < 1$$

$$N(t) = N_0 * 0.5^{\frac{t}{5}}, if\ t \geq 1$$

$N(t)$ is the caffeine concentration in milligrams t hours after the ingestion of N_0 milligrams of caffeine. In our case, N_0 is always 150mg.

For simplicity you MAY$_{59}$ assume that 24 hours after ingestion of 150 mg of caffeine, all caffeine is removed from the body: m 59

$$N(t) = 0, if\ t \geq 24$$

3.7 Consulting The Highscores

Reference: *highscore.html* and *highscore-anon.html*

Available on page: *“Highscores”* (MUST$_{60}$) M 60

3.7.1 Main scenario

1. The application lists the 10 users with the highest caffeine levels for each of the following intervals: the last 24 hours (MUST$_{61}$), the last 7 days (MUST$_{62}$), and since the start of the portal (all time, MUST$_{63}$). For each user, the application displays their rank (MUST$_{64}$), their display name (MUST$_{65}$) which also serves as a link to that user's detailed caffeine statistics (see 3.6), and their highscore caffeine level (MUST$_{66}$). M 61 M 62 M 63 M 64 M 65 M 66

2. The user choses an entry from one of the highscore lists.

3. The application redirects the user to the *“Caffeine Statistics”* page for that entry (MUST$_{67}$). M 67

3.7.2 Exceptions and variants

- 1b. If a user has not made their caffeine statistics public (see 3.2), a placeholder is shown instead
M 68 of their display name (MUST$_{68}$) and no link to their detailed caffeine statistics is provided
M 69 (MUST$_{69}$).

- 1c. If the user viewing the highscores is logged in and has one or more entries in any of the lists,
S 70 the application SHOULD$_{70}$ highlight these entries and append " (You)" to the display name.

3.8 Managing Coffee Kitties

Precondition: The user is logged in

Reference: *kitty.html*

M 71 Available on page: *"Coffee Kitty"* (MUST$_{71}$)

3.8.1 Main scenario

1. The application displays the following:
M 72 — L: A list of coffee kitties the user is a member of, showing their names (MUST$_{72}$) and the
M 73 user's current account balance with that kitty (MUST$_{73}$), as well as a link to the administration
page for that kitty (see 3.9).
M 74 — J: Everything necessary to perform the "Join a Coffee Kitty" use case (see 3.3, MUST$_{74}$)
M 75 — C: Everything necessary to perform the "Create a Coffee Kitty" use case (see 3.4, MUST$_{75}$)

2. The user selects a coffee kitty to administer.

3. The application redirects the user to the administrative page for the chosen coffee kitty.

3.8.2 Exceptions and variants

- 1b. If the user is not an administrator for the coffee kitty, no link to the administrative page must
M 76 be shown and steps 2 and 3 must not be possible (MUST$_{76}$).

3.9 Administering a Coffee Kitty

Precondition: The user is logged in and is the administrator of the coffee kitty.

Reference: *kitty-admin.html*

M 77 Available on page: *"Coffee Kitty"* (via "Administer" link, MUST$_{77}$, see 3.8)

3.9.1 Main scenario

1. The kitty administrator accepts or declines requests to become a member of the coffee kitty (MUST_{78}, see the notes). M 78
2. The application displays a list of the members of the coffee kitty with their full names and display names (MUST_{79}), their current balances (MUST_{80}), and a form field to record payments (MUST_{81}). M 79 M 80 M 81
3. The kitty administrator records payments for each user and submits them.
4. The application stores the data and adjusts the users' account balances (MUST_{82}). M 82
5. The kitty administrator enters the price for one cup of coffee that users of the kitty have to pay (MUST_{83}). M 83
6. The kitty administrator submits the new price.
7. The application stores the new price (MUST_{84}). M 84

3.9.2 Notes

Requests to become a member of a coffee kitty that the current user is an administrator for MUST_{85} be shown in the form of notifications. These notifications MUST_{86} be displayed on every page the user visits and MUST_{87} be shown until the administrator either accepts or declines the request. M 85 M 86 M 87

3.10 User account management

Precondition: The user is logged in.

Reference: *profile.html*

Available on page: variable (see the notes)

3.10.1 Main scenario

1. The application displays the current user's display name and email address in the application's main menu as a link to that user's profile page (MUST_{88}). M 88
2. The user clicks the link to their profile page.
3. The application displays the following information:
   — I: The information submitted during registration (MUST_{89}, see 3.2) M 89
   — D: The user's default coffee kitty (MUST_{90}). M 90
4. The user modifies some or all of the information I (MUST_{91}). M 91
5. The user changes their default coffee kitty D (MUST_{92}). M 92

3.10.2 Notes

S 93 The link to a user's profile page SHOULD$_{93}$ be named like `<displayname> (<email-address>)`.
M 94 In any case it MUST$_{94}$ have the same HTML ID attribute as in the HTML prototype.

M 95 In order to allow changes to the user's details I, the portal MUST$_{95}$ verify the user's authenticity again by asking them for their current password before performing any changes. Also, if the password is to
M 96 be changed, the same new password MUST$_{96}$ be entered twice.

The first coffee kitty that a user successfully joins or creates, automatically becomes their default kitty
M 97 (MUST$_{97}$).

3.11 Resetting the Application

Reference: *index-admin.html*

Precondition: The user is logged in as the application administrator (see 4.4).

M 98 Available on page: *"Home"* (MUST$_{98}$)

3.11.1 Main scenario

M 99
1. After login, the application displays a button or link with which the user can reset the application (MUST$_{99}$).
2. The user resets the application.

M 100
3. The application deletes all persistent data and resets itself to the factory defaults (MUST$_{100}$).

4 Non-functional requirements

4.1 Quality

The quality of a software is determined by a lot of factors. In your solution you MUST$_{101}$ strive to maximize the quality of your solution, in particular: M 101

- correctness and reliability,
- responsiveness,
- scalability (as specified in 4.2),
- robustness,
- security,
- readability (as specified in 4.3),
- overall modularity and maintainability

In the evaluation of your system, these aspects will have far greater weight than any single one of the other requirements. The particular level of quality expected is not quantified, but we expect the functionality of the system to be sufficiently narrow that you could finish a low-quality implementation in a fraction of the available time. Therefore, the expected quality level of your solution is quite high and strong investments in quality will be appropriately rewarded. Please prioritize with your best judgment based on your overall experience.

4.2 Scalability

All of your pages MUST$_{102}$ have a page load time under 2 seconds. Page load time here means the M 102
duration between the time the request is sent by the user's browser and the time the body onload event
is fired or in cases of non-HTML content, the time the first byte of the actual content, ignoring HTTP
headers, is received. Not more than an additional 1.5 seconds MUST$_{103}$ be spent on loading content M 103
asynchronously. During the evaluation we will try to find out how many concurrent requests your
application can handle while still fulfilling these requirements.

4.3 Programming style and documentation

All identifiers and comments in the source code and helper files MUST$_{104}$ be in English. M 104

Each source code file MUST$_{105}$ be documented at least globally (i.e. in its head) and shortly explain its purpose and which other parts of the system use its functionality. M 105

M 106 Each file that you manually create or modify MUST$_{106}$ have one of the following identifiers in a comment at the top of the file:

- `origin: M`, if the file has been created manually from scratch
- `origin: RM`, if the file has been reused but modified.
- `origin: GM`, if the file has originally been generated by some tool and subsequently been modified.

S 107 Each non-trivial public program element (such as a method) SHOULD$_{107}$ be documented (purpose, usage) if this is not clear from its name.

4.4 Persistence and State

M 108 All information (users, coffee kitties, caffeine levels, ...) MUST$_{108}$ be stored in persistent storage and must survive system shutdowns and crashes intact.

M 109 The application MUST$_{109}$ have a default administrative user with email address “organizers@platforms.org” and password “admin”. The only purpose for this user is to be able to reset the application to its factory defaults (see 3.11).

5 Rules for development

5.1 What is allowed

During the contest you may:

- Use any language, tool, middleware, library, framework, and other software you find helpful.
- Reuse any piece of any pre-existing application or any other helpful information you have yourself or can find on the web yourself. Anything that already existed the day before the contest started is acceptable.
- Use any development process you deem useful.
- Ask the organizer (who is acting like a customer) any question you like regarding the requirements and priorities (see 5.4).

5.2 What is not allowed

During the contest you may not:

- Disturb other teams in their work.
- Send contest-related email to people not on your team or transfer the requirements description (or parts thereof) to people not on your team.
- Have people from outside of your team help you. (This includes reusing work products from other teams.) There are two exceptions to this rule: (1) you may use answers of the customer as described in Section 5.4 and (2) you may use user-level preview feedback as described in Section 5.3.

5.3 Intermediate versions and user feedback

During the contest, teams MUST$_{110}$ showcase intermediate versions of their CafMan service to obtain user-level comments and feedback. To do this, host your CafMan service on your development server, open it for public access, and post a notification in the Live Contest Blog as described below. M 110

Users can then comment on your CafMan prototype regarding functionality, defects, usability etc. The teams are allowed to use this user-level feedback for improving their system. They are not allowed to post source code or to use information from outsiders that is on the code level.

We will also use your public preview versions to measure your progress using an automated tool. Your progress will be taken into account during your evaluation, therefore we recommend to publish an intermediate version whenever you finish a unit of work.

5.3.1 How to post a message in the blog

Log in on plat-forms.org. Your username (team2012a, team2012b etc.) and password (a 5-digit number) for the blog is available from Lutz Prechelt or Ulrich Stärk.

Go to Blog.

Middle column: Click "Create new blog entry." above the latest blog post

Fill the "Title" field. Mention the names of your home organization, platform, and the version number of your new prototype, say, "O'Reilly Perl team: Version 6".

Fill the "Body" section. Indicate for which aspects you are particularly interested to receive feedback and perhaps what people should expect from your prototype. Most importantly, provide the URL where to access your prototype.

If you want to provide longer explanations (more than 600 characters), your post will be trimmed and the rest will be hidden behind a "read more..." link. You can affect the place where the cut will be made by putting the HTML comment `<!--break-->` at the desired position. For this purpose, press the "Source" button in the upper right corner of the WYSIWYG editor's menu in order to modify the HTML source manually.

Store your data by clicking "Save". Your post will be published immediately.

If necessary, you can still modify your entry by navigating to it and clicking "Edit". Afterwards "Save" as before.

During the contest, all visitors of www.plat-forms.org can leave a comment on the entry.

5.4 Talking to the customer

The customer of the CafMan project is represented by Ulrich Stärk. He will be more or less available for questions regarding the requirements during most of the day and evening. He will not be available between midnight and 8:00 in the morning.

He will happily answer questions regarding clarification of the meaning of requirements (but beware: if his advice and this document should ever be in conflict, it is this document that is relevant for the evaluation, not his advice). He will only vaguely answer questions regarding requirements priorities or regarding which of two concrete solution ideas he would prefer and will point you to this document instead. He will not look at your solutions or give concrete feedback about them.

5.5 Delivery

You are allowed to finish your development at any time you think appropriate, but no later than the end of the contest at 18:00 on 2012-10-10. Early delivery will be taken into account during the evaluation.

Package and submit your deliverables as follows:

1. Create a file named `adminuser.txt` containing two lines: the first line must be the login name of a user with unlimitied administrative rights for your virtual machine's operating system, the second line must be that user's password. We MUST$_{111}$ be able to log in to your virtual machine using those credentials. M 111

2. Shut down the virtual machine that is running your CafMan. Put all files that make up the virtual machine and the `adminuser.txt` file into a ZIP file called `vmware.zip`. The virtual machine MUST$_{112}$ be set up in such a way that your CafMan automatically starts up when the machine is booted and that your CafMan does not need manual shutdown when the machine is shutdown. (The machine will always get at least 10 seconds of idle time before shutdown.) M 112

3. Package a snapshot of your versioning database into a ZIP file. The versioning database is the subtree in file system of your versioning server that contains the directories and files holding all versions of each file of your CafMan project you have under version control. If you used Git as your versioning system, call the ZIP file `git.zip`; if you used Subversion, call the ZIP file `svn.zip`; etc.

4. Create a source code distribution of your CafMan implementation and package it in a ZIP file called `cafman-sources.zip`. This distribution should contain all files needed to recreate an instance of your CafMan except those that already existed before the contest and were not modified. So you should include all source code, build files, skripts, configuration files, documentation, etc, if they are new or were modified. You need not include pre-existing infrastructure such as application server, database server, compiler, libraries, unmodified parts of frameworks etc. The content of this ZIP file wil be considered published under the OpenSource license(s) you specified when you requested participation in the contest.

5. Create a ZIP file named like <homeorganization>.zip, e.g. accenture.zip. This ZIP file contains the three other ZIP files mentioned above and is your deliverable (MUST$_{113}$). M 113

6. Determine the 160-bit SHA-1, 256-bit SHA-256, or 512-bit SHA-512 message digest checksum of the deliverable. We will call this checksum the "fingerprint" of the deliverable. SHA-{1,256,512} is computed for instance by the sha{1,256,512}sum utility from GNU coreutils.

7. Send a two-line email containing the name of the deliverable and the fingerprint to organizers@platforms.org. The reception time timestamp of this email will be your submission time and MUST$_{114}$ be before the end of the contest. Use the name of the deliverable as the subject of the email. The organizers will ignore all but the last such email from each team. M 114

8. Fill in the 12 questions in the file postmortem-questionnaire.rtf. We need a separate questionnaire from each member of your team. Since you may not have enough energy left to provide

sensible answers today, you need not do this right now; it is OK if you send the filled-in questionnaire at any time until Friday next week (2012-10-20). However, the information you give us in the questionnaire is important input for the scientific evaluation, so we kindly ask that each team member thoroughly provides his/her own personal answers. Please send the result to organizers@plat-forms.org. Thanks!

9. Create a medium containing the deliverable. Write the name of the deliverable and the time of day on it with a pen. Hand the medium over to the customer (Ulrich Stärk). YOU ARE DONE. Come to our get-together, go and sleep or party or bang your head against a soft wall — whatever you feel most like doing. Thank you veeery much for participating in Plat_Forms!

M 115 **The fingerprint sent in your email absolutely** MUST$_{115}$ **match that of the ZIP file on the medium or else your whole participation was in vain.** The fingerprint email has two purposes: (1) to make the submission timestamp independent of your medium-creation progress and (2) to validate the medium, in particular a replacement medium should the original one turn out to be unreadable.