CUSTOMER REQUIREMENTS

EVERYTHING PROGRAMMERS NEED TO KNOW BEFORE WRITING CODE

BY MARCO BEHLER

THANKS FOR READING

Thanks for reading *Customer Requirements - Everything Programmers Need To Know Before Writing Code.*

If you have any suggestions, feedback (good or bad) then please do not hesitate to contact me directly at marco@marcobehler.com or leave a comment on our blog at www.marcobehler.com/blog.

(This is a one-man operation, please respect the time and effort that went into this book. If you came by a free copy and find it useful, you can compensate me at http://www.marcobehler.com/books)

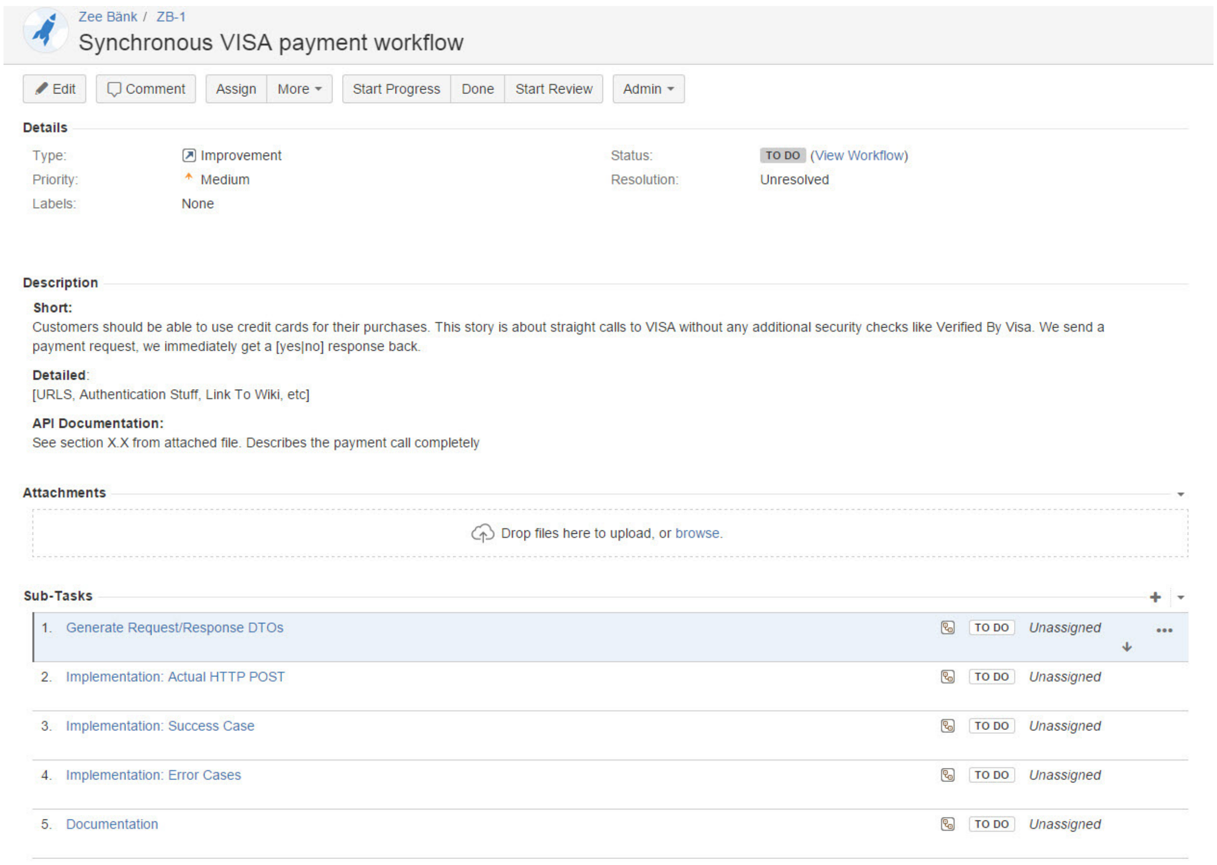
Thanks!

[illegible]

- Marco Behler, Author

Practice Time: Estimation & Billing – From gut feelings to a time tracking database

Back to our start-up Zee Bänk. In the Communication chapter we came up with 5 tasks for our synchronous VISA payment workflow:


Zee Bänk / ZB-1

Synchronous VISA payment workflow

Details

Type:	Improvement	Status:	TO DO (View Workflow)
Priority:	Medium	Resolution:	Unresolved
Labels:	None		

Description

Short:
Customers should be able to use credit cards for their purchases. This story is about straight calls to VISA without any additional security checks like Verified By Visa. We send a payment request, we immediately get a [yes|no] response back.

Detailed:
[URLS, Authentication Stuff, Link To Wiki, etc]

API Documentation:
See section X.X from attached file. Describes the payment call completely

Attachments

Drop files here to upload, or browse.

Sub-Tasks

1.	Generate Request/Response DTOs	TO DO	Unassigned
2.	Implementation: Actual HTTP POST	TO DO	Unassigned
3.	Implementation: Success Case	TO DO	Unassigned
4.	Implementation: Error Cases	TO DO	Unassigned
5.	Documentation	TO DO	Unassigned

1) Generate Request/Response DTOs. *Estimate = ?*
2) Implementation: Actual POST. *Estimate = ?*
3) Implementation: Success Case. *Estimate = ?*
4) Implementation: Error Cases. *Estimate = ?*
5) Documentation. *Estimate = ?*

Our goal is to have an hour estimate for each one of those tasks.

Estimation Techniques

Before looking at each one individually, I like to quickly go through all the tasks and decide on what my strategy for each task is going to be like:

Gut Feeling + Deviation Factor

Generating Request/Response DTOs I have done plenty of times before, so I trust my guest estimate and my own deviation factor of 1.5. I assume I start with a blank time tracking database for now, so I cannot really look up old, similar tasks. But for sure I can write my new estimates down.

The Success and Error cases I have already reasoned about a lot and as I assume that there are no real technical challenges, I will also go with my gut feeling + estimation factor.

What I should note here is that my gut feeling always also includes time for testing and a tiny bit of cleaning up, not just writing the code itself.

Spiking

The Actual POST is something I am not quite sure of. Sure, it seems simple enough and the documentation seems to be up-to-date. But I want to get a quick feel it is really as simple as it looks, if the parameters are right, if authentication is easy. My spike is going to consist of a couple of simple CURL calls. Nothing too fancy and maybe using up 30 minutes.

Fixed Time-Limits

The documentation I am not going to estimate, instead I am going to set a fixed time-limit of 1 hour. The one hour is simply a measure from experience. Maybe I can copy & paste parts of existing documentation for documenting this new feature, maybe I have proper writing tools in place, maybe I do not. Your number might differ.

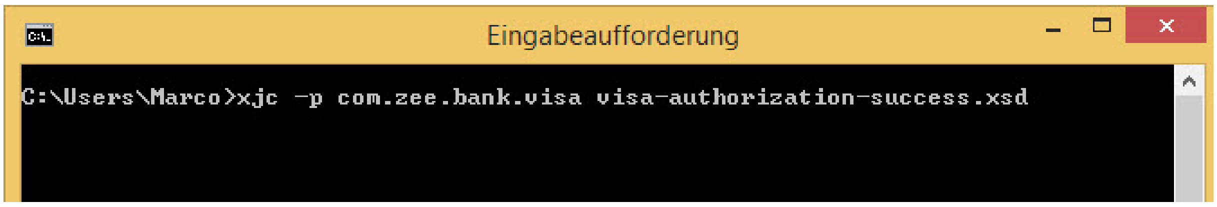
Writing documentation is in some ways a lot like brushing your teeth: Skipping to brush your teeth for the whole month and then on one day brushing 31 times is not really going to work. Same for documentation. Do it in small batches and do it regularly.

The Nitty-Gritty

I recommend you to quickly go back to the Line-Of-Thinking sections from the Communication Practice chapter and read through them again, as there is a certain overlap.

Generate Request/Response DTOs (maybe automatically from XSD)

For the non-Java users out there: Tools exist to easily create .class (e.g. XJC) files from XSDs and in our case VISA provides all XSD location URLs in our mock documentation file. So our spike consists of invoking a couple of command line calls, maybe cleaning up the result, adding it to our VCS etc. Time for the gut feeling:



```
C:\Users\Marco>xjc -p com.zee.bank.visa visa-authorization-success.xsd
```

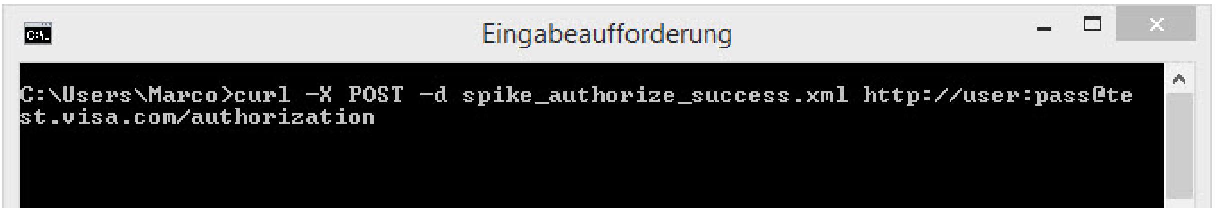
My Final Estimate: 30 minutes x 1.5 (deviation factor) = 45minutes.

Implementation: Actual POST

Let us assume that after curling the VISA test servers I have a pretty high confidence level that everything is in place, as expected. So we are talking about getting our favourite HTTP client like Apache HttpClient up and running and implementing a POST. Marshalling/Un-Marshalling request/response objects can be a bit of a pain to setup at first, but let us assume it is easy in this case. Other than that, POSTs are straightforward and there is plenty of documentation available how to do a straight post.

For Socket/Read Timeouts I decide to go the easy way: If the request to VISA takes longer than 3s, I timeout and return an error code. (This is rather simplified to how it works in the real world, but will do just fine for now). I also am going to have a couple of tests for the timeouts etc.

Time for the gut feeling:



```
C:\Users\Marco>curl -X POST -d spike_authorize_success.xml http://user:pass@te
st.visa.com/authorization
```

My Final Estimate: 1.5h x 1.5 (deviation factor) = 2,25h.

Implementation: Success Case

For the success case we assumed the redirection logic and merchant notification system etc. to be already in place. So the success case basically consists of a couple of lines of implementation hooking into an already existing codebase and a few tests for that.

My Final Estimate: 1h x 1.5 (deviation factor) = 1.5h.

Implementation: Error Cases

Error cases are always trickier than the success cases. Remember, we identified 3 error cases: WRONG CVC, BLOCKED, EXPIRED. We have different acceptance criteria for each error state: Sometimes we prompt the user to try his purchase one more time, another time we show the user a simple “contact-customer-support” error message. We assume the redirection and general error message logic (i.e. display an error message on a respectively styled CSS page) to be already in place.

We have to read in some property files for those error messages, react upon the error code VISA returns us and then either retry the purchase or display a final error message. And tests.

My Final Estimate: (3x2,5h) x 1.5h(deviation factor) = 11,25h

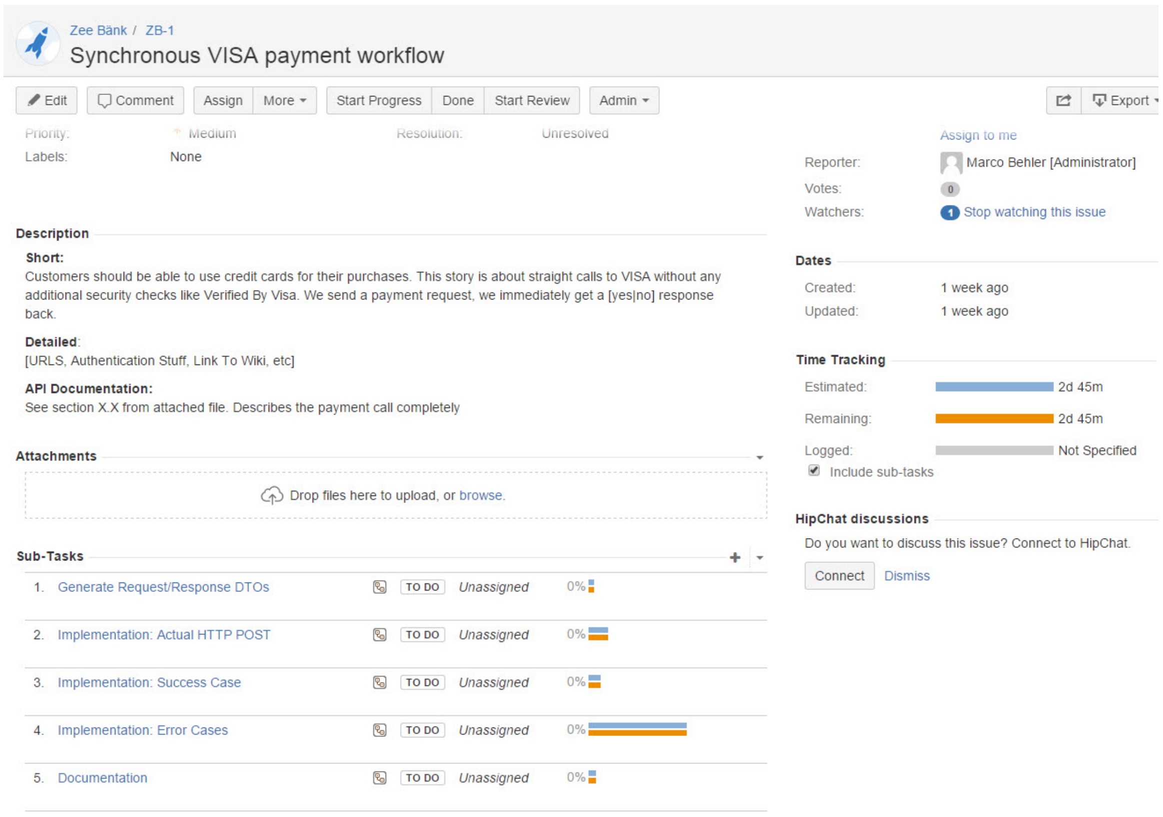
Documentation:

My Final Estimate: 1hr (fixed)

The Result

Adding up all these numbers leads to 16,75h or roughly 2-3 days for the synchronous VISA payment task. That number might differ for you depending on how experienced you are. It does not matter. We are going to note down the estimates, later on look at the actual implementation time and then adjust accordingly in the future.

What we can see by looking at the screenshot below, is that the error task is assumed to be a lot more time-consuming than the other ones. I still advise against splitting it up into more sub-tasks though. It is faster to let one programmer implement the whole thing than trying to parallelise it with multiple programmers and more sub-tasks. The communication overhead is simply too big for that.



Synchronous VISA payment workflow

Priority:	Medium	Resolution:	Unresolved
Labels:	None		

	Assign to me
Reporter:	Marco Behler [Administrator]
Votes:	0
Watchers:	1 Stop watching this issue

Description

Short:
Customers should be able to use credit cards for their purchases. This story is about straight calls to VISA without any additional security checks like Verified By Visa. We send a payment request, we immediately get a [yes|no] response back.

Detailed:
[URLS, Authentication Stuff, Link To Wiki, etc]

API Documentation:
See section X.X from attached file. Describes the payment call completely

Dates

Created:	1 week ago
Updated:	1 week ago

Time Tracking

Estimated:	2d 45m
Remaining:	2d 45m
Logged:	Not Specified

☑ Include sub-tasks

Attachments

Drop files here to upload, or browse.

HipChat discussions

Do you want to discuss this issue? Connect to HipChat.

Connect Dismiss

Sub-Tasks

1.	Generate Request/Response DTOs	TO DO	Unassigned	0%
2.	Implementation: Actual HTTP POST	TO DO	Unassigned	0%
3.	Implementation: Success Case	TO DO	Unassigned	0%
4.	Implementation: Error Cases	TO DO	Unassigned	0%
5.	Documentation	TO DO	Unassigned	0%

Recommendations

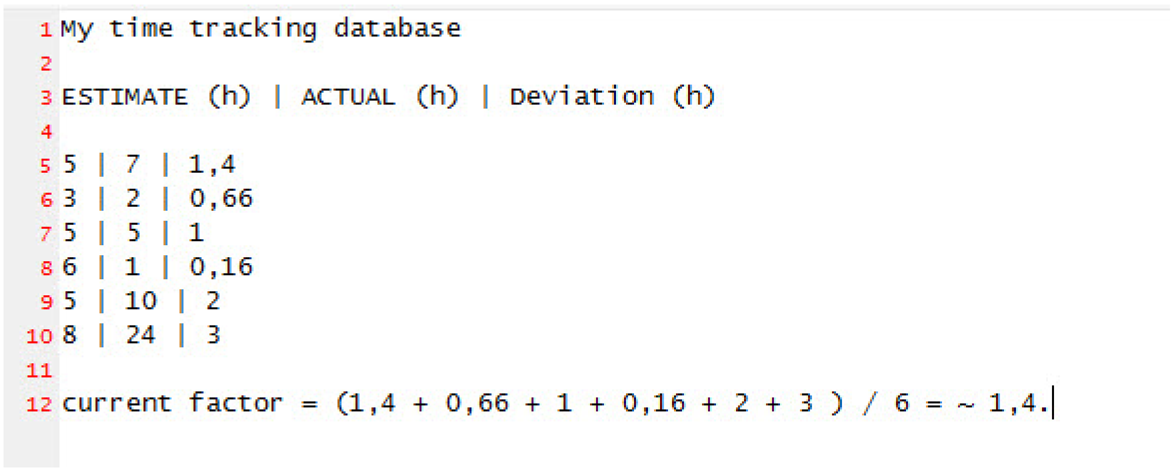
Now that we have estimated our tasks, let us look at how we can get our estimates more reliable over the long-term.

A simple time tracking database

If you live in poor man's land and doubt the whole time-tracking thing, you can start off with just this. Use Excel or Notepad or something similar. Do not even write down the task descriptions, but whenever you tackle a new task or user story write down a quick hourly estimate in a new line.

Use a time tracker if possible, otherwise, if you are really lazy, just note down how long your task took to implement from your head. Look at the deviation. Do that for every task you implement for a month. Note down your deviation factor.

Here is what this looks like in Notepad:



```
My time tracking database

ESTIMATE (h) | ACTUAL (h) | Deviation (h)

5 | 7 | 1,4
3 | 2 | 0,66
5 | 5 | 1
6 | 1 | 0,16
5 | 10 | 2
8 | 24 | 3

current factor = (1,4 + 0,66 + 1 + 0,16 + 2 + 3 ) / 6 = ~ 1,4.
```

According to that file my estimates are off by 1.4 the real effort and I am going to use this number for my next estimates and update it again after a couple of weeks. Yes, this is not a perfect number and I do not want to get into a discussion on averages, means and standard deviations here. For practical purposes, this will be more than enough.

-- END OF PREVIEW --