Programmer Search Plan (v1.2)

U	Understand: Do I have a clear view of (or even understand) what I want to search for?
P	(Create) Plan: What am I going to search for and how? Which Search terms? Which search engine?
E	Execute Plan: Use the right search machine, select, read
A	Adjust: mention your sources, search and adjust so you understand the result
V	Validate Result: Did I find what I was looking for? Is the problem solved?
L	Learn/Look Back: Knowing the outcome, could I have done it smarter? Next time: what would I do different?

For more go to https://github.com/coentjo/softwarelessons

Programmeur ZoekSchema (NL) (v1.2)

U	Understand. Begrijp: Snap ik het probleem waarvoor ik een oplossing zoek?
P	(Create) Plan. Maak Plan. Waar ga ik naar zoeken? Hoe? Welke zoektermen? Welke zoekmachine?
E	Execute Plan. Voer Plan uit: Gebruik de zoekmachine volgens plan, selecteer en lees.
A	Adjust: vernoem je bronnen, zoek uit en pas aan zodat je het resultaat snapt.
V	Validate Result. Controleer resultaat. Gevonden wat ik zocht? Probleem opgelost?
L	Learn/Look Back. Leer: Nu ik het antwoord weet, zie ik nu wellicht handiger manieren voor een volgende keer?

For more go to https://github.com/coentjo/softwarelessons

Contents

1 Example use **1**
1.1 Decimal to Roman 1

2 What is this document about? **2**

3 How to use this as a programmer/student? **2**

4 How to use this as a coach/lecturer? **3**

5 Is there more? And can I share it? **3**

1 Example use

1.1 Decimal to Roman

Case description: create a program that translate a number between 1 and 100 to Roman notation[1].

Understand: After reading the problem we first thought we understood the problem, but writing out some examples the problem at hand appeared to be a little more difficult. For example 23 becomes XXIII and 16 becomes XVI. Looks like you can translate the digits of the decimal number one by one and glue them together: the '2' in 23 becomes XX and behind it we write the III which comes from the '3'. Even when a number ends in 4 or 9 (a I is placed before V or X) this works: 24 becomes XXIV and 29 becomes XXIX (or would that be IXXX?: google).

This would mean the *algorithm* would (for numbers up till 99) consist of translating the *ten*s and then the *one*s. But think, there are exceptions to that: 49 becomes IL instead of XXXXIX, and 99 becomes IC.

(Create) Plan: Our plan was to first check for the exceptions, and for every other case break up the number in two digits (adding a '0' if the number has only 1 digit), translate both digits and put the results together.

Execute Plan: Having this *algorithm* written on the white board the students were

[1] or, if that seems too simple, go up to 1000.

assigned the task to realise it in C#. They later told me they succeeded, so I was not involved in the latest steps.

Validate Result:

Learn/Look Back:

Reflection: That morning I introduced them to the first version of the Search Scheme. When it became apparent the assignment of the Roman numbers was hard for the students I reminded them at the Scheme. Afterwards I was convinced at first this was a good example of how to use the scheme. Later came the insight that the scheme was used here as general problem solving scheme, not as the search scheme it was meant to be. One way to look at this is that it is beautiful to see that the search scheme resembles a general problem solving scheme, maybe we don't need to differentiate between those schemes. That would be elegant! But I wás searching for an example use of the scheme and did not find it yet.

2 What is this document about?

When learning how to program you learn software concepts. When creating software you're always searching for a way to translate the features you have to build or the problems that arise to those concepts. Often this means searching the internet for solutions other programmers have already thought of. This document is about how to search for a good solution to your problem. When working on this concept it proved useful to have a web site providing more info and examples: see the link at the bottom of this page. Probably the website is a more dynamic place to be.

3 How to use this as a programmer/student?

Understanding the Problem

Starting point: Error? Exception? Bug? Feature on a wishlist?

Do I understand what I need to know to get going?

Do I know the meaning of relevant terms and conceptas?[2]

[2]for example when my starting point is a compiler error: do I understand all the words in the error and

Create Plan

What to search for and how?

Execute the Plan

Check Result

Is my problem solved? If not go back to '*Understand*' to see if your understanding of the problem was correct, carry on from there to 'P' and so on.

Look back and learn

What did you learn from your actions?

Looking back: do you see a better/faster/smarter way to approve the same result? Did you search for the right thing? When you encounter the same problem, maybe you already know what to do then? Would it be wise to store some info?

4 How to use this as a coach/lecturer?

Lecturers in programming courses could use the scheme by providing it to students and refering to it when a search is needed. The scheme started as a variation of the scheme proposed in the classic book 'How To Solve It' by Polya that helps solving a math problem.

5 Is there more? And can I share it?

This document is a work in progress. In the footer of every page you can find a link to the newest version! It is published to be used, I hope it will help students (and their teachers) at learning how to search the solution for programmer problems. If you have any additions, remarks, questions, ideas or other experiences that you can mail me (`mailto:C.Crombach@Fontys.nl`).

 This document is created in LaTeX. I'm

do I know the concepts? If not it may be wise to search for that first.

thinking about putting the sources in public some time, let me know if you are interested! If you want to have the sources for the website you are also encouraged to contact me. The website was built in Seaside (Pharo).