# Circle Language Spec Strategy

*Author: JJ van Zon*

*Location: Oosterhout, The Netherlands*

*Date: February 2008 – January 2020*

## Contents

Contents 1

Introduction 1

Higher Goal & Doubts 1

Earlier Goal 2

Later Goal 2

Theme Picking 4

Focus on Easy Themes 4

Redo Easy Themes 4

Rules 4

Plan the Specifics 5

Project Order 5

Emphasized Again 5

Specific Limitations 5

Do not cover *uses* 5

Productive Writing 6

Writing Efficiently 6

Standard Work in Next Projects 9

Conceptual Explanations vs Diagram Explanations 10

## Introduction

This document describes the strategy, goals and the approach of the project *New Computer Language*.

*The New Computer Language* is about ideas about a new programming language, mostly about a diagram notation that shows things, that go on inside a computer. There are other ideas in it about programming language in general, but those are not as prominent.

This document explains *a lot* of details of how the approach taken in the work. The project was split up into many sub-projects.

## Higher Goal & Doubts

One of my worries is that I have not learned enough constructs that exist in the world to really make a unified language for all digital objects. The language was supposed to give a different view onto any digital data: folders, scheduled tasks, object oriented code, the web, databases or any complicated referential structure. It is supposed to combine all of it into a single landscape in a single language, and serve as an entry-point for all other types of digital expression, such as opening data in the app it is intended for, displaying that view on top of the diagram, that is a map of everything going on inside the computer.

**Goal of the Language**

Here are some goals of the new language:

- Gain instant insight into complex referential structures hopefully to a degree that no other way of diagramming ever could.

- *Anything* digital can be translated to such a diagram, because most things digital are just referential structures.

- Make it easier to program software.

- Make *programming* the same as *using* a computer.

- Offer an instant user interface for anything digital and traditionally invisible.

- The invisibility of digital connections, that software developers and ICT infrastructure engineers only imagine in their head, are now visualized on screen.

What might prove that the diagram language fails:

- That the diagram expression does not simplify the view on complex referential structures, but that it will still look like an unoverviewable bunch of wiring.

- Do take comfort in the fact, that if this goal is not reached, the language might still have other uses. It might still be a nice way to navigate and write your code.

The language has more goals, but the goals above might be the more important ones.

## Earlier Goal

A while ago the goal of the project was to describe my ideas adequately for others to be able to pick up work where I left off, if that would ever happen. This was done by making a functional design on paper of the new computer language, adequate to turn the computer programming language into a usable product. It was meant to preserve my thoughts. Then I could either sell the idea, give it away, implement it myself or not work on it for years and then the ideas will not fade away. It will give me peace of mind and it will open up possibilities for me and for the idea.

Were the descriptions of the language to be adequate, I would not have to worry about whether I work on it anymore, because the ideas would be preserved.

However, later in the project, most of that goal was already realized.

## Later Goal

Later, the goals for the new computer language went beyond describing my existing ideas adequately.

The goals of the sub-projects would no longer be just to describe an element of the system.

The goals of the sub-projects would lean more towards turning the new computer language into a finished and usable product.

Product is a loose term here. It can mean a working piece of software in which you can use the new computer language. But a product could also be a specification on paper of what the new computer language would look like.

Some stuff got finished.

Some things were put off till later.

New things were imported into the project.

Priorities got changed.

Goals got redefined.

There could be multiple intermediate goals. Each goal would be about taking the language to the next level.

- Documentation: Adequate OO Paradigm (mostly finished)

- Documentation: Automatic Diagram Organization (part finished)

- Programming: Program a version (part done)

- Business: Release

- Documentation: Added Paradigm: Concepts

- Documentation: Added Paradigm: Querying

- Documentation: Advanced input/output

These phases are bigsteps. Each of them is a big step.

The 'release' thing, is most important to me now (2020-01), and the rest of it is not.

The first phase is 'Documentation: Adequate OO Paradigm'. It only consists of a couple of sub-projects. It is mostly done. The end result is pretty much a full specification of an object oriented diagram language in which object oriented systems can be expressed in a large amount of detail (almost the fullest detail).

The second phase is 'Documentation: Automatic Diagram Organization' is only part done. The thing is: when diagrams do not organize themselves automatically (for instance positioning of te shapes and curving of the lines), the effectivity of the language might be dramatically decreased, to a point that the intentions with the language might not be reached. If the way diagrams organize themselves is automatically worked out, the usuability of the language might be very much increased.

Programming: This was the point at which, it was found a good plan to actually program a version. Even though a lot of the language would not have been designed yet. Some experimental things have been programmed. One fear was: making something, that would later be thrown away, would the language change completely. But the base of the language was not expected to change dramatically. Another fear was to lose time programming, that was better spent on the language design. The hope was, that a working version would tell you where the big problems are would indicate solutions to problems you initially could not get your head around.

At this point it might actually be released it to the public. I used to want to patent it, but that is no longer the case. Open sourcing it is what I am going for. I don't see myself making money off of this and it is no use just sitting on the idea having time just pass by.

That is where it's at right now. The other steps in the list above are possible future steps, but not necessarily.

## Theme Picking

This section covers strategies for rough planning and *theme picking*. The most important thing is to focus on the easiest stuff. This is not just lazy. It has advantages.

### Focus on Easy Themes

Inadequately finished-up documentation was moved to the bottom of the documentation folders. Only if they are better finished up, they might be moved back to the top of the documentation again.

First describe things that are already clear ideas. First skip the harder themes, on which the view is not clear yet. There are many themes, that can be relatively easily worked out. It is quicker and that work is more 'overdue'.

One reason for this is to get more work done quicker. Another reason is, that the current projects are about making *existing* ideas easier to pick up by others, instead of covering *new* ideas. Another argument is: if you document harder topics, you will create documentation of lesser quality, less easily accessible. So the right decision seems to be to do easier topics first.

### Redo Easy Themes

Some of the more recently done documentation is still tough to read.

This is because when writing the documentation, the concept was still being worked out. The documentation was written in a way easy to write. It was tough enough getting a concept straight on paper to begin with. You can not blame anyone or anything for the material first to be written in a way in which it is easier to *write*. But later, the material and the reading order should change, so it is easier to *read*.

Hazards that might make the reader loose confidence in you:

- Too much

- Too difficult

- Too much junk in it

- Not finished = not worth reading

Some chapters cover things with an approach, that is too difficult and too tough on the reader.

### Rules

- Focus on themes easy to work out is a strict rule.

- Simplify existing chapters.

- Each theme only a couple of weeks.

- Use *themes* as smallest unit for time planning / progress monitoring.

- No separate progress monitoring within sub-projects.

- Quickly make documenting existing ideas a past stage.

- Documenting existing ideas = most important.

- Do not plan for the difficult topics: the difficult topics might become easier in the future.

### Plan the Specifics

- Which easy subjects *exactly?*

- Which existing topics to simplify *exactly?*

- Which tough subjects *exactly?*

- Define all projects

- Make rough planning

- 8 easy subjects: 2 months

- Simplifying existing topics: 1 month

- Tougher subjects: how long?

Plan for easy topics only. You do not know how difficult the harder topics will be to you in the future.

### Project Order

This paragraph may be short, but this is quite important. This is the planned order in which to do things:

**- Easy subjects**

**- Simplify existing subjects**

**- Tougher subjects**

(perhaps by the time you get to them, they will be easier)

### Emphasized Again

The idea now is to work out the ideas that are already clear, so that the existing ideas are easier to pick up by others, instead of working out concepts, that I have not worked out well in my head yet. This should be quicker and easier.

### Specific Limitations

Now details will be covered, less important to the approach, but lengthier in text.

#### Do not cover *uses*

I used to have the idea, that for the interfaces theme of the new computer language, I would read about design patterns. Reading about design patterns at this stage, might actually be a bad idea for progress.

The reason why considering design patterns would be harmful is: if you do it for the *Interfaces* articles, you would read it to orient yourself in the different uses of interfaces, in order to explain those in that documentation section. But I set the rule, that I would not cover all the different uses of a concept, but just the blunt description of the concept. Sometimes uses add necessary explanations to a documentation section, but the rule is to not cover all the different uses. The implementation of interfaces in the new language can be bluntly explained, and it should not raise much discussion among programmers if they already know how interfaces work in other languages.

Also: design patterns might introduce new concepts, that would make me want to revise large parts of the language. That would be harmful to progress. I had better work out existing ideas the way they are now, instead of changing them all again and having no straight forward story to tell.

So those things would make reading about design patterns at this stage a bad idea.

## Productive Writing

In contrast to the previous section, this section is not about rough theme picking, but more about strategies for when you have picked a theme and start to work on that theme.

This section covers mostly strategies about the details of writing, but also other project execution strategies.

### Writing Efficiently

I used to have too many demands for the articles, that I produce.

It is a big project and it was going really slowly.

I was putting a lot of extra effort into make the articles perfect.

But I decided I should be focusing more on making the articles *adequate*.

I may need to get used to them looking less perfect from now on.

There are several kinds of tasks involved in projects for writing these articles:

- Collect existing ideas

- Brainstorm about systematics

- Organize & reformulate brainstorm texts

- Determine article list

- Converting brainstorm texts to articles

- Start over

- Use old content as a sort of cross out list.

- Write articles

- Brainstorm about different approach

- Adapt articles to different approach

- Add references to other articles

- Process details

- Reading over

- Folder organization

- Scatter texts from old documents across a new form of documentation subdivision

It might be that I was too perfectionistic.

The goal of this documentation should to be kept clearly in mind.

The side-issues can be ignored.

I might have to repeat this general goal in every project, that I start.

I can not have a side-goal just for the fun of it, because each extra hour seems one hour too much.

- Creative vs. productive:

- Form a clear idea in your head.

- It has to be clear in your head,  
before it can be made clear on paper.

- Your whole conciousness may need to understand the concept.

- You are not a production machine.

- You can not make this, if you focus on producing articles and do not even understand the language anymore.

- If I have totally forgotten the workings of an important concept, that I wrote about before, then I have been focusing too much on producing the documentation, rather than forming the idea in my head.

Below, each task is commented on.

The focus lies on article content, not on making things perfect.

- Collect existing ideas

It is quite important to collect all existing ideas before starting a project.

That does not change, even though it is a lot of work.

- Organize & reformulate brainstorm texts

I think, the way I organize and reformulate brainstorm texts is just fine, but sometimes it is better to just derive a topic list from the idea texts and start over.

- Creative thinking

Brainstorming about systematics is not going to get easier. I should not see it as production work. You have to do creative thinking and take time to make it crystal clear.

Do not stick to the frame of the project.

It is *one* programming language. You need to keep the bigger picture in mind, even though a project is to work out everthing about a specific subject, for instance: *interfaces*.

Always look at the broader view too.

- Determine article list

I focus too much on explaining each concept in its own separate article.

That is too *form-*oriented.

I try to turn the work into countable reliable items.

I realize too little, that I am going to have to do *creative* thinking much of the time.

I see thin­­gs too much as production work, or look for the easiest way to get the set of articles done.

I have to focus more on *creative* thinking, instead of *productive* thinking.

Split up the material into less articles, if that makes it easier to finish the project.

Make it a single article, for all I care.

Sometimes, when things are too complicated, I ‘artificially’ split up the story into articles. But that is only done, so that I get a clearer view on things so then it is alright to do that.

What I am also noticing, is that I used to make the article list, a list of concrete tangible concepts. Sometimes there are rules, that apply to multiple concepts then the rules were repeated over and over again in multiple articles. I should allow myself to isolate a rule into a separate article and not repeat it everywhere.

- Converting brainstorms to articles

You have to be more satisfied with having a set of stories, the way they were in the brainstorm texts. Do not give the articles an entirely different subdivision, than the brainstorm texts have.

- Article content

**defend**:

When I write articles, I tend to *defend* the system too much.

**compare**:

I try to *compare* the system too much to other systems.

**how & why**:

I talk about too many because’s and *how* and *why*.

**blunt**:

Be more *blunt* about how things work.

**level or knowledge of the reader**:

I worry too much about the *level or knowledge of the reader*.

The material does not have to be read by any person or by an average programmer.

I should not explain the workings of the CPU, the working of other languages, etcetera.

Only if I have been confused about something myself, I may need to explain it, for myself.

I cover *general* notational rules too much in *specific* diagram articles.

The notational rules are covered separately.

I do not need to refer to them.

I worry too much about the implications of *automatic containment*.

**exceptional cases**:

I should stop worrying about *exceptional cases*.

**implications**:

I should not be paranoid about not considering the *implications* of things enough.

**uses**:

Stop explaining all the uses of the language.

You will not cover them all.

You need to provide basic building blocks, not examples of what to do with it.

**delete**:

Do not be afraid to permantly delete texts that fall under any of the categories above.

I am paranoid about the details, but they will come to light during the *implementation* of the system.

I impose too many rules, that I must later break.

I put too much thought into rules, that I cannot uphold.

The list above may work well as a set of rules for the ‘Reach Goal’ part of a project, as opposed to the ‘Make Perfect’ part of the project. But perhaps you should not cover details at all, because they also burden the reader.

- Start over

Sometimes when you have a huge collection of ideas and brainstorms it works better to just extract a topic list out it and start over completely.

- Cross out

Later you can use older material as a cross-out list, to delete older stuff already covered by a newer story and delete stuff that is no longer interesting and to extract stuff that might still be interesting.

- Adapt articles to different approach

I might want to put *more* time in this, than I currently do.

Adapting articles to a different approach is a lot of work.

But I seem to think of it as production work, but I might not want to see it that way.

You should not focus on *getting it over with*.

You should be explaining the new *concept* well and getting the details right in your head.

Sometimes I leave in facts, that just might not be right, just so I can get things over with.

That ruins the story.

- Article form

Too many hours go into adding references to other articles everywhere.

Too much time goes into worrying whether everything I refer to has already been explained or not.

Too much time goes into giving the article a perfect form.

- Processing details

At the end of the project I have a list of details. First consider not to adapt the articles to those details at all.

- Reading over

I read over articles too much.

At the end of a project, consider not to read over all articles. It might take too much time.

- Folder organization

- Simplification in folder organization is good.

- But do not worry too much about ‘it’s easier for the reader this way’ and stuff. Too much time goes into being too friendly to the reader.

In the story above, I am actually also trying to break things up into logical units too much. That is too form-oriented.

### Standard Work in Next Projects

In each next project you pick, however time-costly, *do* process the idea box / loose ideas you might have. Reformulation of loose ideas might actually result in a readable, structured article covering the topic. That is one of the strategies that might actually lead to good text.

Each project, in which you update documentation, could be trailed by an update to the redirection pages, that tell something about that documentation.

### Conceptual Explanations vs Diagram Explanations

Conceptual explanation and the demonstration of the diagram notation were separated in the past. This was done, because sometimes you can conceptually explain something without encountering any problems, and the problem may only be how to express it. In that case it is easy to first draw out a conceptual point of view, and that makes it easier to work out a diagram notation. The conceptual point of view then does not change, but an imperfect diagram notation may be changed later, when your vision on it becomes clearer. Also it allows you to put everything into the context of one single language, even when not every concept has its own diagram notation yet. Also: sometimes when you work out the conceptual explanation first, it becomes easier to figure out a proper diagram notation for it. However, you might later want to merge conceptual explanation with diagram notation explanation in certain cases, because this is often easier on the reader.