# Circle Language Spec Strategy

*Author: JJ van Zon*

*Location: Oosterhout, The Netherlands*

*Date: February 2008 – April 2020*

## Contents

Contents 1

Introduction 1

Earlier Goal 1

Later Goal 2

Theme Picking 3

Focus on Easy Themes 3

Redo Easy Themes 3

Rules 4

Plan the Specifics 4

Project Order 4

Emphasized Again 4

Productive Writing 4

Writing Efficiently 5

Standard Work in Next Projects 8

Conceptual Explanations vs Diagram Explanations 9

Do not cover uses 9

## Introduction

This document describes the strategy and goals of the project *Circle Language Spec*.

*Circle Language* is an idea for a new computer programming language, mostly about a diagram notation that shows things, that go on inside a computer. It might become a way to navigate and write code.

This document quite extensively explains the details about how the work was done. The project was split up into multiple sub-projects to keep things manageable.

The document mostly talks about strategy of work already done. But the same strategies might be of use later.

## Earlier Goal

A while ago the goal of the project was to describe ideas locked up the head. It was meant to preserve the thoughts about this language. This was done, by attempting to make a specification of the new programming language, adequate as a starting point for turning it into a usable product. Then the idea might be sold, given away, implemented or not worked on for years and then the ideas would be less likely to fade away. Theoretically if anyone wanted to, they might be able to continue working on it from that point on. That might create peace of mind and possibilities for the idea.

Later in the project, that goal was mostly realized already.

## Later Goal

Later the goals for the new programming language went further than describing the existing ideas adequately.

The goals of the sub-projects would no longer be just to describe *each element* of the system separately. The goals of the sub-projects would lean more towards turning the new programming 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 programming language. But a product could also be a specification on paper of what the new programming language would look like.

Some things were finished.

Some things were put off until later.

New things were imported into the project.

Priorities were changed.

Goals were redefined.

There could be multiple intermediate goals. Each goal could be about taking the language a step further.

- Documentation: Adequate OO Paradigm (mostly finished)

- Documentation: Automatic Diagram Organization (part finished)

- Business: Release

These phases are bigsteps.

The 'release' thing, is most important now (2020-04), and the rest of it not really.

The first phase was 'Documentation: Adequate OO Paradigm'. It consisted of a limited set of sub-projects. It is mostly done. The end result would be a more or less complete specification of an object oriented diagram language in which object oriented systems can be expressed in a certain amount of detail.

The second phase is 'Documentation: Automatic Diagram Organization'. It was only part done. The thing is: when diagrams do not organize themselves automatically (for instance positioning of the shapes and curving of the lines), the effectivity of the language might be reduced, possibly to a point that the intentions with the language might not be reached as much. If the diagrams would organize themselves automatically, the usability of the language might increase.

The next phase might be to release it to the public. The goal used to be to patent it, but that is no longer the case. Open sourcing is the goal now. Not sure how to make money off of this, and it seems no use just sitting on the idea and have time just pass by.

## Theme Picking

This section covers strategies for rough planning and *theme picking*. The most important point may be to focus on the easiest stuff. This is not just lazy. There are advantages.

### Focusing on Easy Themes

Less finished-up documentation was moved to the bottom of the documentation folders. If a topic is more complete and polished, it might be moved back to the top of the documentation again.

The idea is to first describe things, that are already clear ideas, first skipping the harder themes, with less of a clear view. There are multiple themes, that can be worked out with more ease than others. That may be quicker and that work might be more 'overdue'.

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

### Redoing Easy Themes

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

This might be, because when writing the documentation, the concept was still being formed. The documentation may have been written in a way easy to write. Sometimes it was tough enough getting a concept straight on paper. You might not be able to 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 might be changed, so it will become easier to *read*.

Hazards that might make the reader loose confidence in you:

- Too much

- Too difficult

- Too messy

- Not finished = not worth reading?

But that might be stated a bit unforgivingly.

### Rules of Thumb

- Perhaps try to focus on themes easy to work out.

- Perhaps give more priority to existing chapters.

- Each theme: only a couple of weeks, is the aim.

- An idea might be to use *themes* as smallest unit for time planning and progress monitoring.

- Maybe do not do separate progress monitoring for the smaller parts.

- Maybe try to make documenting existing ideas a past stage sooner than later.

- Documenting existing ideas seems most important

- Maybe do not plan for the difficult topics; the difficult topics might become easier in the future.

### Plan the Specifics

- Perhaps it is a good idea to define more specifically:

- Which easy subjects*?*

- Which existing topics to simplify*?*

- Which tough subjects*?*

- Perhaps define a list of intended future sub-projects

- Maybe make a rough time planning

- 8 easy subjects: 2 months

- Simplifying existing topics: 1 month

- Tougher subjects: how long?

Maybe plan for easy topics only. The idea is: you might not know how difficult the harder topics will be in the future. They may become easier to define as time passes by and after basic topics are worked out successively.

### Project Order

This paragraph may be short, but this might be important for some focus. As a rule of thumb this is the planned order in which to do things:

**- Easier subjects**

**- Simplify existing subjects**

**- Tougher subjects**

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

### Emphasized Again

The idea is to work out the ideas that are already clear, so that the existing ideas are easier to pick up, instead of working out concepts, that are not crystalized out very well yet. This may be quicker and more easy.

## Productive Writing

In contrast to the previous section, this section is not so much about theme picking, but more about possible strategies for when you have picked a theme and want to start working on that theme.

This section covers mostly strategies about writing texts, but also some other project execution strategies.

### Writing Efficiently

There used to be quite some demands on the articles to produce.

The project was not small and it was going sort of slowly.

Effort went into make the articles 'perfect'.

But then the decision was made, to try to focus more, on making the articles *adequate*.

A thing to get used to might be, to have them look less 'perfect' from now on.

There can be several kinds of tasks involved in the projects for writing these articles:

- Collecting existing ideas.

- Brainstorming about systematics.

- Organizing & reformulating brainstorm texts.

- Determining article lists.

- Converting brainstorm texts to articles.

- Starting over.

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

- Writing articles.

- Brainstorming about different approach.

- Adapting articles to different approach.

- Adding references to other articles.

- Processing details.

- Reading over.

- Folder organization.

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

The project may have been executed a bit too perfectionistically.

The goal of the Circle Language Spec could be kept in mind more clearly.

You might want to ignore the side-issues.

This general goal could be repeated at the start of each sub-project.

Maybe side-goals cannot be in there just for the fun of it, because each extra hour seems one hour too much, when things are going slowly.

- Creative vs. productive:

- Try to form a clear idea in your head.

- You might need to have it clear in your head,  
before you can attempt to can make it clear on paper.

- Your whole consciousness may need to understand the concept first.

- You might not want to see yourself as a production machine.

- You might not be able to make this, if you only focus on producing articles and not so much on understanding the concept.

- When you have totally forgotten the workings of an important concept, written about before, then maybe the focus has been too much on producing documentation, rather than forming the idea in your head.

Below, each task is commented on.

The focus lies on article content, not on perfection.

- Collecting existing ideas

You might give some importance to collecting all existing ideas (notes, 'idea boxes') at the beginning of a project. That does not change, even though it might be quiet some work. The collection of those ideas may actually be a good starting point, for the eventual content. You might miss out on something good, if you do not do it.

- Organizing & reformulating brainstorm texts

The method employed for organizing and reformulating brainstorm texts seems ok, but sometimes it might be better to just derive a topic list from the idea texts and start over.

- Creative thinking

Brainstorming about systematics might not ever become easier. You might not want to see this as production work. You may want to do creative thinking and take some time for it to become crystal clear.

It may be a good idea, to not just stick to the frame of the project. It is *one* programming language. You may want to keep the bigger picture in mind, even though a project is to work out everything about a specific subject, for instance: *interfaces*. It should be a good idea to look at the broader view too.

- Determining article list

Sometimes there was too much focus on explaining each concept in its own separate article.

That may be too *form-*oriented.

It may have happened in an effort, to turn the work into countable reliable items.

There may have been realized too little, that it might be better to have the focus on *creative* thinking most of the time.

Perhaps work was considered production work at times, or looking for the easiest way to get the set of articles done.

It might be better to focus more on *creative* thinking, instead of *productive* thinking.

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

Make it a single article, if you must.

Sometimes, when things are too complicated, you might ‘artificially’ split up the story into articles. This can be done, to get a clearer view on things, not so much because it reads better, or because it is required.

Sometimes effort went into making the article list a set of concrete tangible concepts, which is not always required. Sometimes when there were rules, that apply to multiple concepts, the rules were repeated in multiple articles. It should be allowed to isolate a rule into a separate article and to not repeat it.

- Converting brainstorms to articles

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

- Article content

**defend**:

Maybe try not to *defend* the system too much.

**compare**:

You might not want to *compare* the system too much to other systems.

**how & why**:

Maybe try to avoid talking about too many because’s and *how* and *why*.

**blunt**:

Maybe tend to be more *blunt* about how things work.

**level or knowledge of the reader**:

Perhaps try not to worry too much about the *level or knowledge of the reader*.

The material does not have to be read by everybody. It can be jargon.

It does not have to explain the workings of the CPU, the working of other languages, etcetera.

If, when writing, you experience confusion about something, you may want to explain it, for yourself.

*General* notational rules do not have to be covered in *specific* diagram articles. Maybe you think it is useful for clarity's sake. But those notational rules could be covered separately. Rules also do not necessarily need to be referred to.

**exceptional cases**:

Maybe try not to worry about *exceptional cases*.

**implications**:

Maybe try not to worry about not considering the *implications* of things enough.

**uses**:

Perhaps try to avoid explaining all the different uses of the language. It would be hard to cover them all, since it is a general purpose programming language. The basic building blocks can be provided. It might not be necessary to give all the examples of what to do with it.

**delete**:

Perhaps try not to be afraid to permanently delete texts, that fall under any of the categories above.

Maybe try not to worry about the details. They might resolve naturally later, perhaps when programming an application for it. A side-effect of covering much detail, might be imposing rules, that you would later break. Or that effort goes into coming up with rules, that are hard to uphold.

The list above may work well as a set of rules for a sort of 'prototyping' part of a project, as opposed to a 'finishing touches' part of a project. But perhaps it is wise to not cover details, because that also might make it easier for the reader.

- Starting over

Sometimes when you have a big collection of ideas and brainstorms, it may work better, to just extract a topic list out of it and start over.

- Cross out

Later you might use older material as a cross-out list, to delete older things already covered by a newer story and perhaps delete things that have become less interesting and extract stuff that might still be interesting.

- Adapting articles to different approach

Adapting articles to a different approach may take a while.

But sometimes putting more effort into this might be good.

You may be tempted to see this as production work, but you might not want to see it that way.

Maybe try not to focus on *getting it over with*.

Maybe try to explain the new *concept* well and getting the details right in your head.  
Sometimes facts seem left in, that just might not be right, just to get it over with.

That may leave the story in a lesser state.

- Article form

Too much effort might go into adding references to other articles everywhere.

Too much effort might go into worrying, whether everything referred to, has already been explained or not.

Too much effort might go into giving the article a 'perfect' form.

- Processing details

At the end of the project there might be a list of details left. You first might want to consider, not to adapt the articles to those details at all.

- Reading over

Consider to not read over the articles as much.

At the end of a project, consider not to read over all articles. It might take time you would rather not spend.

- Folder organization

- Simplification in folder organization seems good.

- But maybe do not worry too much about ‘it’s easier for the reader this way’ and stuff. Now might not be the time, to accommodate the reader as much, given you have so much documentation to produce.

The story above, might actually also try to break things up into logical units too much. That might be too form-oriented. Sometimes the rules seem to contradict. They're just guidelines.

### Standard Work in Next Projects

In each next project you pick, however time-costly, *do* consider processing 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 a diagram. In that case it can be easier 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.

### Do Not Cover Uses

There used to be the idea, that for the interfaces theme of the new programming language, it would be good to read more about design patterns. But reading about design patterns at that stage, might not actually have been a good idea for progress.

The reason why considering design patterns would be less good 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 one of the 'rules' was not to 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 was to not try to cover all the different uses. The implementation of interfaces in the new language could be bluntly explained, and it may not raise much discussion among programmers, if they already know how interfaces work in other languages.

Also: design patterns might introduce new concepts, that might inspire to revise large parts of the language. That might be harmful to progress. It might be better to 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 might make reading about design patterns at this stage not such a good idea.