



# Software Architecture Documentation in the Real World

---

**Markus Völter**

voelter@acm.org

www.voelter.de



# C O N T E N T S

- What is Software Architecture
- Documenting Software Architectures
  - (Structured) Glossaries
  - Patterns and the Pattern Form
  - Pattern Languages
  - Tutorials and FAQs
  - Diagramming and Modeling
  - Channels
  - What about Code?
- **Layout and Typography**
- Diagramming Guidelines
- Summary



## Page Layout & Typography

- Typography influences the reader when reading the document
- You'll read faster if the **page geometry** is suitable and you've chosen **suitable fonts**
- You should use **document templates**
  - that contain only stylistic aspects, not 25 sections to fill in
  - They are prepared by a small number of people
  - Hence, good layout will become pervasive
- And always use **change marks** for revisions of the documents – otherwise readers will not read anything beyond version 1



## Page Layout & Typography II

- **50% Page contents**

- seems to be too little
- but is appropriate for the readers' fields of view
- Typically a good decision for documents



- **2 – 2.5 Alphabets per Line**

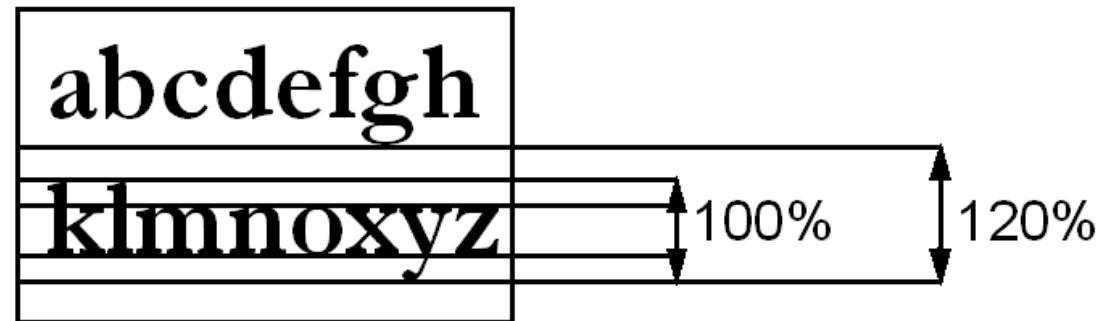
- Long lines are **hard to follow**
- Short lines require **too many “carriage returns”**
- Might result in **several columns** in a document

abcdefghijklmnopqrstvwxyz abcdefghijklmnopqrstvwxyz



## Page Layout & Typography III

- 120% Line Spacing



- 2 Fonts

- Use Serif Font for the text (guides the eye)
- Use Sans Serif for Headlines
- ... and maybe Monospaced for Code

### Beispiel

Für die Überschrift wurde hier die serifenlose Schrift Frutiger gewählt. Im Haupttext kommt die Serifen-Schrift Garamond zum Einsatz.

### Beispiel

Für die Überschrift wurde hier die serifenlose Schrift Helvetica gewählt. Im Haupttext kommt die Serifen-Schrift Times New Roman zum Einsatz.



## Page Layout & Typography IV

- **Use Variations Carefully**

- CAPITALS require 12% more reading time!
- Italics and Bold is more suitable
- Do not use underlines – ugly!

shape

*shape*

**SHAPE**

- **Max 3 levels of structure**

- Chapters, Sections, Subsections
- Things like 4.1.2.3.4.5 are not useful

- **Use graphical gimmicks** (lines, symbols), but use them sparsely



## Page Layout & Typography V

- **Enough Whitespace around illustrations**
  - Make sure illustrations are not jammed in between text
  - Use a different (Sans Serif) font for captions
- **Line Width for Illustrations**
  - Make sure the line width of illustrations is compatible with the weight of the font in the running text
  - Otherwise the illustration will disrupt the layout of the page
- **Spelling is important!**
  - ... correct grammar and readable wording is important, too!
  - Short, simple sentences are better.
  - Consider the document literature! Write a book!
- **Use Active Voice!**
  - Talk to the reader: it is easier and more engaging to read!



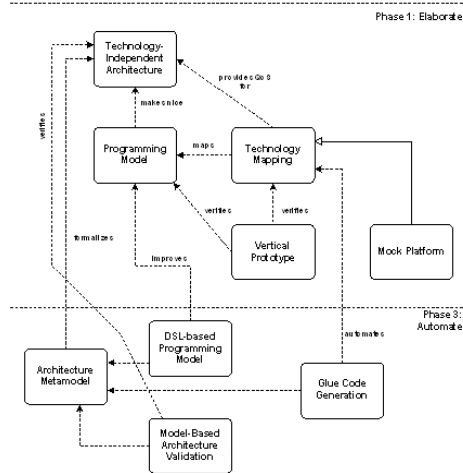
# Page Layout & Typography VI (Line Width for Illustr.)

## Bad:

VERTICAL PROTOTYPE. It is important that you do this *before* you dive into phase 3: Automation.

**Automation:** The third phase aims at automating some of the steps defined in the first, and refined in the second phase, making the architecture useful for larger projects and teams. First, you will typically want to GENERATE BLUE CODE to automate the TECHNOLOGY MAPPING. Also, you often notice that even the PROGRAMMING MODEL involves some tedious repetitive implementation steps that could be expressed more briefly with a DSL-BASED PROGRAMMING MODEL. Finally, MODEL-BASED ARCHITECTURE VERIFICATION helps ensure that the architecture is used "correctly" even in large teams.

The following illustration shows the patterns and their dependencies. It uses well-known UML dependency and inheritance notation.



## Good:

### Known Uses

All MDSB projects that I am or was involved in have used this approach, this includes a C-based component model for embedded real time systems, web applications and components for mobile devices.

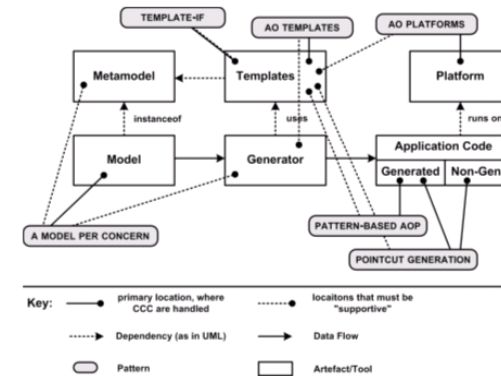
The documentation of the openArchitectureWare generator [OAW] shows an extensive practical example of using more than one model as generator input.

### Summary

In non-trivial scenarios, A MODEL PER CONCERN is absolutely necessary to keep (large) models manageable. Make sure you use a tool where this approach can be implemented painlessly, before you use the generator tool on larger projects.

### Pattern Overview – Pt.2

The following illustration shows where in an MDSB infrastructure the respective CCC-handling approach will take effect. For example, AO TEMPLATES handle the cross-cutting concerns in the templates, while the generator tool has to support it by providing the AOP support for template files.



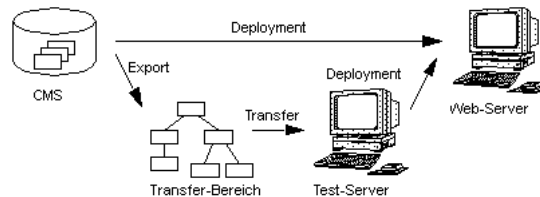
This section provides a summary of the consequences in the form of a chart. The more grey in the box, the better. The rationale for the length of the bars is derived from the consequences sections of the respective patterns.



# Examples

## Content-Deployment

Im Projekt unterscheiden wir zwischen zwei Verfahren, Änderungen am Content des Portals vorzunehmen. Redaktionelle Änderungen werden von den Redakteuren des Portals durchgeführt. Strukturelle Änderungen am Content erfordern Programmierung und Tests und werden nur vom Projekt-Team vorgenommen.



## Redaktionelle Änderungen

1. Ein Redakteur erstellt neue Dokumente im CMS oder ändert bestehende Dokumente.
2. Ein Chefredakteur gibt die Änderungen frei und publiziert die neuen Dokumente. Der neue Content wird unmittelbar im Portal sichtbar.

## Strukturelle Änderungen

1. Ein Programmierer ändert die JSPs der entsprechenden Templates.
2. Nach Abschluss von Programmierung und Modul-Test werden die Templates vom CMS in einen speziellen Transfer-Bereich exportiert.
3. Von dort werden die Templates auf den Test-Server übertragen. Hier findet der System-Test statt.
4. Die Schritte 1 bis 3 werden bei Bedarf wiederholt bis der Systemtest erfolgreich verläuft.
5. Ein Administrator spielt die Templates des Test-Servers auf dem Web-Server der Produktionsumgebung ein und startet den Web-Server neu.

## Who should read this paper?

This paper is intended to be read by software architects (as well as consultants, coaches and developers), who work in medium to large sized project teams. For the stereotypical three-person-project many of the patterns will probably be considered overkill. Also, the patterns described below are probably most useful in projects that build platforms, large, long-lived systems or in the context of product-line architectures.

## Introduction

Why write a paper on software architecture? There are several reasons. The most important is that I think the craft of software architecture in current industrial practice is not what it should be.

Before I start bashing current practice, I want to state what this paper is actually about. I think, there is a difference between the functional architecture of a system, and the technical architecture. The functional architecture is aligned with the domain. For example, it is about understanding processes, responsibilities, variabilities; in one word it's about what the system should *do*. Technical architecture on the other hand is about how the functional architecture is implemented: do we have components? Are we distributed? How do we scale? What about systems management? How do we realize the required QoS? How are processes rendered? Do we use a relational or a non-relational DB? In this paper, I focus primarily on technical architecture. Specifically, I want to show, how we can come up with a technical architecture that makes the development of the functional architecture (i.e. the realization of the use cases for the system) as pain-free as possible.

## Why software architecture is important

Software architecture has been, is, and will be an important discipline in software development. At some point, you have to come up with a consistent metaphor for how your system is structured and behaves. There are different opinions on *when* you have to define your architecture (at the beginning of a project, or on the fly), *who* should do that (one or more architects, the development team as a whole), *how detailed* it should be defined (just a rough spec or detailed prescriptions) and *in what way* to specify it (powerpoints, word docs, code snippets, metamodels).

Also, in some circles, the word architecture itself has accumulated so much negative connotation, that it is not used at all: people use terms such as "strategic design" instead.

However, I think it is agreed that a non-trivial system has to stick to certain consistency rules internally in order to communicate its internal structure to (new)



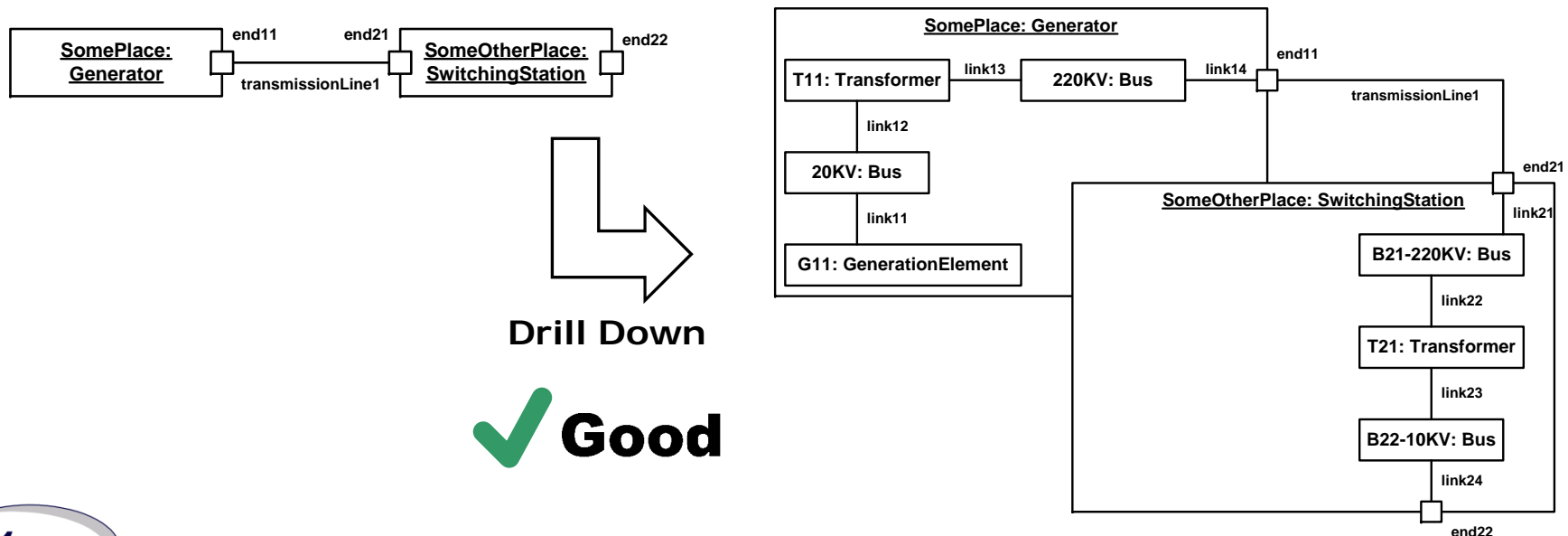
# C O N T E N T S

- What is Software Architecture
- Documenting Software Architectures
  - (Structured) Glossaries
  - Patterns and the Pattern Form
  - Pattern Languages
  - Tutorials and FAQs
  - Diagramming and Modeling
  - Channels
  - What about Code?
- Layout and Typography
- **Diagramming Guidelines**
- Summary



# Diagramming Guidelines

- **Limited Real Estate**
  - Diagram should be viewable on a screen
  - printable on a sheet of paper (Letter, DIN-A4)
  - $7 \pm 2$  boxes/entities
- **Hierarchical Decomposition** (with Drill-Down diagram)
  - Make sure all elements in a specific diagram are the same level in the hierarchy





## Diagramming Guidelines II

- **Always explain diagrams**, the picture itself is not enough
  - Give it a half-sentence **title**
  - **Explain** in prose **what the diagram shows** (or use the diagram to illustrate concepts explained in the running text)
  - In the explanation **don't explain every detail** (parameters, eg.) shown in the diagram, but help people "find their way" around the diagram
- Provide a **diagram key** (generally: well-defined language)
  - A diagram is only useful if readers can know **what a graphical element means** (boxes and lines do need explanation!)
  - Hence, either provide a **key**, or use a **well-known language** for the diagram



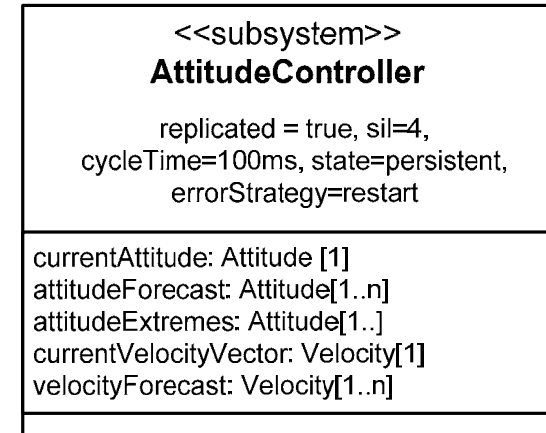
## Diagramming Guidelines III

- **Clearly defined “message”**
  - A diagram should have a **well-defined purpose**,
  - Hence, it should typically only **illustrate one concern**, aspect, viewpoint, abstraction level or layer in a hierarchy, relationship kind, ...
  - ... unless it's purpose is to explicitly **illustrate the relationships** of some of these concerns, viewpoints or aspects
- **Readable Left-to-Right or Top-to-Bottom**
  - (most) People naturally scan a diagram from **left to right**, or from **top to bottom**
  - Layout your diagram so it can be read in these orders
  - Especially important if there's some kind of **signal flow**, **time progression** or **increasing level of detail**

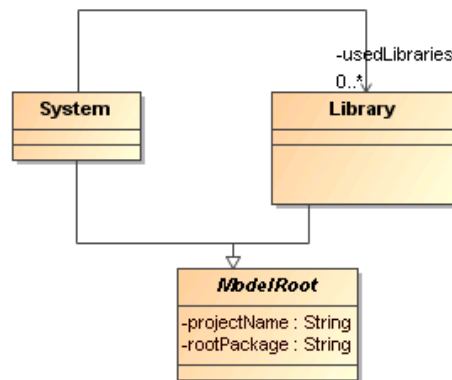


## Diagramming Guidelines IV

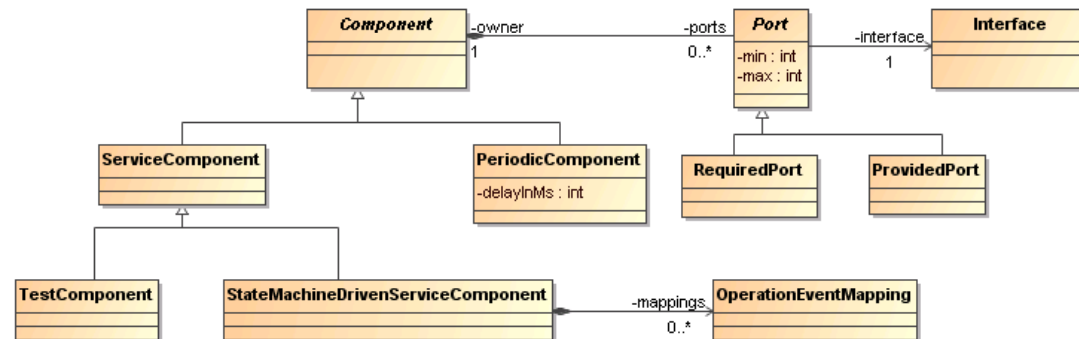
- **Don't add too much text to diagrams**
  - Rather, add these details to separate views, property lists, or render them as graphical elements
- If possible, run the **same kind of relationship** in the **same direction**
  - E.g. inheritance vertical, associations horizontal, dependencies diagonal



**X Bad**



**X Bad**



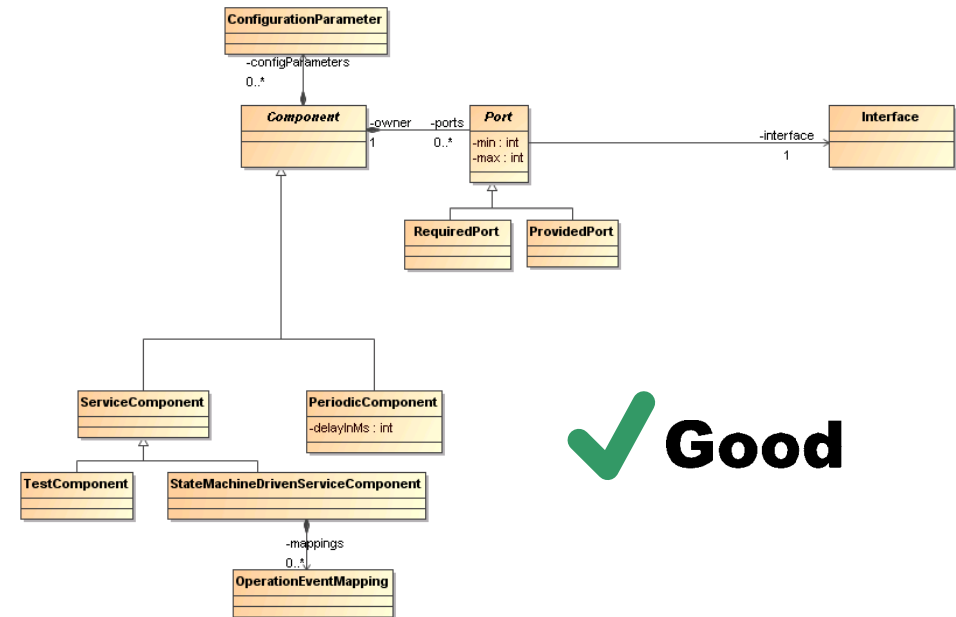
**✓ Good**



# Diagramming Guidelines V

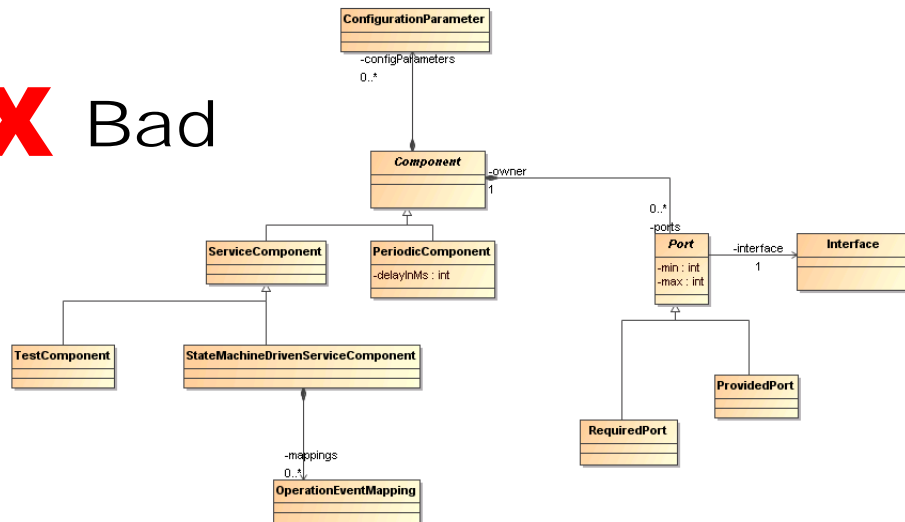
- Graphical Proximity has meaning

- Cohesion
- Grouping



**✓ Good**

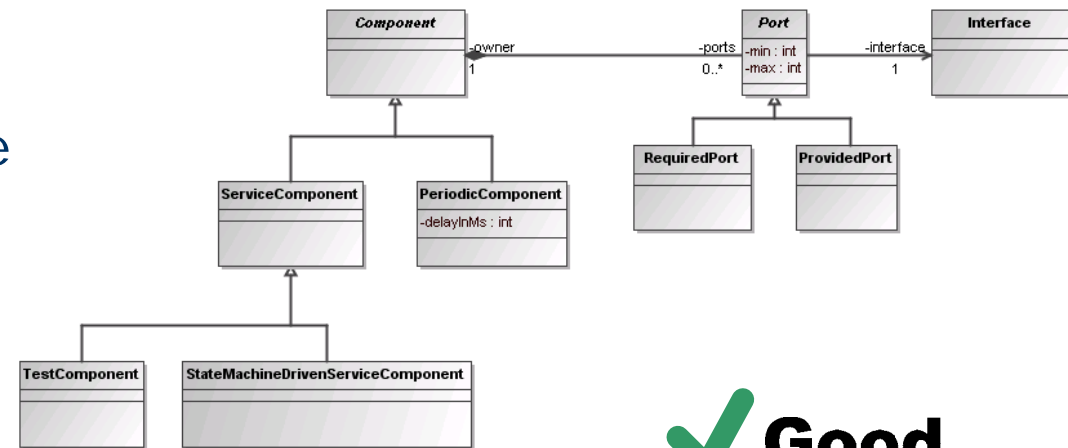
**X Bad**



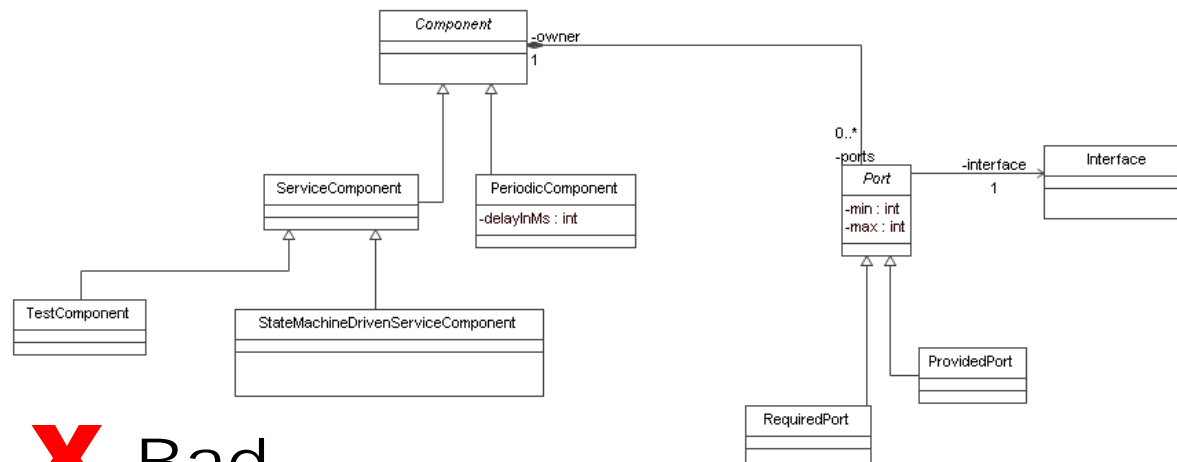


## Diagramming Guidelines VI

- **Make it generally nice**
  - As few lines as possible (join/fork lines)
  - Join lines if possible
  - Line Width, Fill Color
  - Use a drawing tool, not a modeling tool!



**✓ Good**

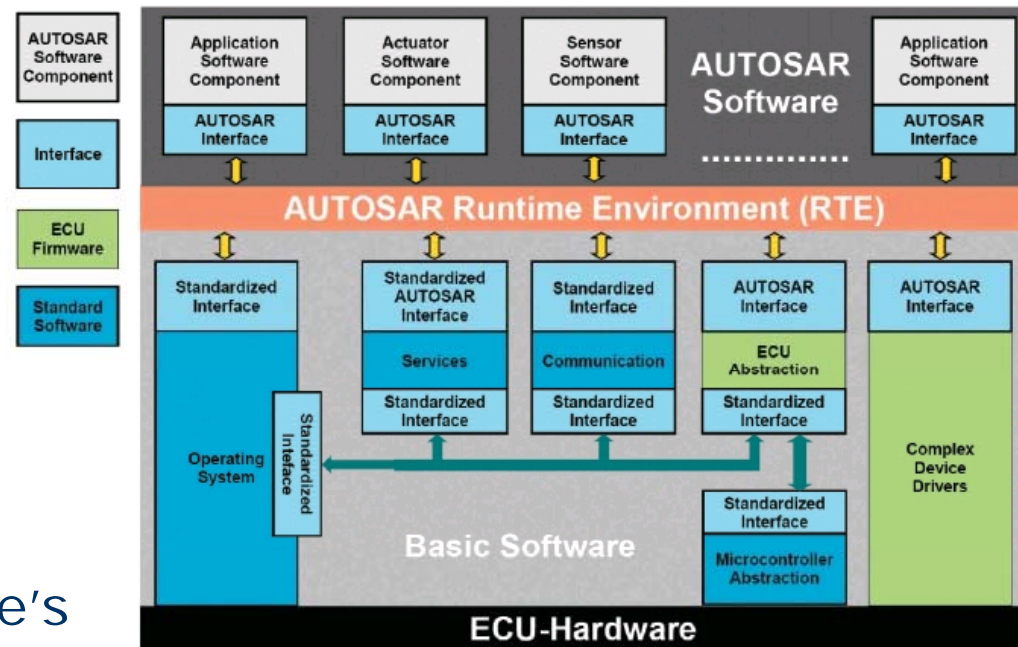


**X Bad**



## Diagramming Guidelines VII

- Don't imply stuff you don't mean to say
  - Layers are a good candidate...
- Use few colors
  - Every color should have a defined meaning
  - It is part of the language's concrete syntax



✓ Good ? X Bad

Is this a layered architecture?



## Diagramming Guidelines VIII

- And finally ... don't force diagrams.
- Use diagrams for **what they are good for!**
  - Relationships between things
  - Processing steps (with in/out parameters)
  - Timelines
  - Signal Flow
  - Causality
- There are other ways of rendering things:
  - Tables/Matrices
  - Textual Notations