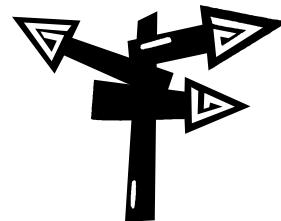


Design Guidelines for Domain Specific Languages





Gabor Karsai², Holger Krahn¹,
Claas Pinkernell¹, Bernhard Rumpe¹,
Martin Schindler¹, Steven Völkel¹

1. Department of Computer Science 3 (Software Engineering)
RWTH Aachen University, <http://www.se-rwth.de/>
2. Institute for Software Integrated Systems
Vanderbilt University Nashville, USA


Outline

- | | |
|----|-----------------------|
| 1. | Introduction |
| 2. | DSL Design Guidelines |
| 3. | Discussion |
| 4. | Conclusion |





Design Guidelines for Domain Specific Languages



1. Introduction



Martin Schindler
Software Engineering
RWTH Aachen
University
Page 4

Why Design Guidelines for DSLs?

- Designing a new DSL
 - needs **experience** and
 - is sometimes error-prone and **time consuming**

- Existing **tools** simplify **technical aspects** but lacks support for a **good language design**

- Guidelines based on
 - our experience in developing languages
 - relying on existing guidelines on general purpose and modeling languages

⇒ **Target:**
Support DSL developers to achieve better **quality** of the language design and **acceptance** among its users.

Martin Schindler
Software Engineering
RWTH Aachen
University
Page 5

Categories of DSL Design Guidelines


- 5 categories along the language development process:
 1. **Language Purpose:**
What is the aim of the language?
 2. **Language Realization:**
How to implement the language?
 3. **Language Content:**
Which elements should be included?
 4. **Concrete Syntax:**
How to define a readable representation of the elements?
 5. **Abstract Syntax:**
How should the language be represented internally?

SE SOFTWARE ENGINEERING

RWTH AACHEN

Design Guidelines for Domain Specific Languages

2. DSL Design Guidelines



<p>Martin Schindler Software Engineering RWTH Aachen University Page 7</p>	<h2>1. Language Purpose</h2>
<ul style="list-style-type: none"> ▪ Guideline 1: <i>"Identify language uses early."</i> <ul style="list-style-type: none"> • many forms of usage: documentation, analysis, configuration, code generation, ... • differences strongly influence needed language concepts ▪ Guideline 2: <i>"Ask questions."</i> <ul style="list-style-type: none"> • Who is going to model in the DSL? • Who is going to review the models? When? • Who is using the models for which purpose? ➔ Identify the domain, its experts, and the development process ▪ Guideline 3: <i>"Make you language consistent."</i> <ul style="list-style-type: none"> • DSLs are typically designed for a specific purpose • each feature of a language should contribute to this purpose 	

<p>Martin Schindler Software Engineering RWTH Aachen University Page 8</p>	<h2>2. Language Realization</h2>
<ul style="list-style-type: none"> ▪ Guideline 4: <i>"Decide carefully whether to use graphical or textual realization."</i> <ul style="list-style-type: none"> • both approaches have advantages/disadvantages • weight and match against end users' preferences and uses ▪ Guideline 5: <i>"Compose existing languages where possible."</i> <ul style="list-style-type: none"> • by embedding, using extendable languages, or referencing • concepts of the composed languages need to fit together ▪ Guideline 6: <i>"Reuse existing language definitions."</i> <ul style="list-style-type: none"> • by language extension or language specialization • or taking existing definitions as a starter ("language pattern") ▪ Guideline 7: <i>"Reuse existing type systems."</i> <ul style="list-style-type: none"> • improves comprehensibility and avoids misinterpretations 	

<p>Martin Schindler Software Engineering RWTH Aachen University Page 9</p>	<h3>3. Language Content (1/2)</h3> <ul style="list-style-type: none"> ▪ Guideline 8: <i>"Reflect only the necessary domain concepts."</i> <ul style="list-style-type: none"> • by validating the language definition against the domain (e.g. using examples) • to ensure expressiveness for all necessary domain concepts ▪ Guideline 9: <i>"Keep it simple."</i> <ul style="list-style-type: none"> • one of the main targets • eases implementation, introduction, understandability, ... • achieved by guidelines 10-12 ▪ Guideline 10: <i>"Avoid unnecessary generality."</i> <ul style="list-style-type: none"> • by preventing generalization or parameterization not yet needed
--	---

<p>Martin Schindler Software Engineering RWTH Aachen University Page 10</p>	<h3>3. Language Content (2/2)</h3> <ul style="list-style-type: none"> ▪ Guideline 11: <i>"Limit the number of language elements."</i> <ul style="list-style-type: none"> • sublanguages can cover different aspects of the system/domain • libraries extend expressiveness based on basic language elements ▪ Guideline 12: <i>"Avoid conceptual redundancy."</i> <ul style="list-style-type: none"> • concepts with none or slightly differences are often source of confusion ▪ Guideline 13: <i>"Avoid inefficient language elements."</i> <ul style="list-style-type: none"> • efficiency of a model should be transparent to the language user → should not depend on specific elements used within the model
---	---

<p>Martin Schindler Software Engineering RWTH Aachen University Page 11</p>	<h2>4. Concrete Syntax (1/3)</h2>
<ul style="list-style-type: none"> ▪ Guideline 14: <i>"Adopt existing notations domain experts use."</i> <ul style="list-style-type: none"> • inventing a new concrete syntax raises the barrier for domain experts → chose syntax close to existing notations (within the domain or other common used languages) ▪ Guideline 15: <i>"Use descriptive notations."</i> <ul style="list-style-type: none"> • supports learnability and comprehensibility ▪ Guideline 16: <i>"Make elements distinguishable."</i> <ul style="list-style-type: none"> • basic requirement to support understandability • usually a document is written only once but read many times → efficiency for the reader more important than for the writer 	

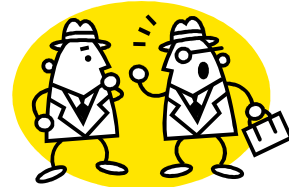
<p>Martin Schindler Software Engineering RWTH Aachen University Page 12</p>	<h2>4. Concrete Syntax (2/3)</h2>
<ul style="list-style-type: none"> ▪ Guideline 17: <i>"Use syntactic sugar appropriately."</i> <ul style="list-style-type: none"> • syntactic sugar improves readability • but an overuse can hide the important content ▪ Guideline 18: <i>"Permit comments."</i> <ul style="list-style-type: none"> • enables explanation of design decisions • for better understanding or even documentation ▪ Guideline 19: <i>"Provide organizational structures for models."</i> <ul style="list-style-type: none"> • possibility to arrange models in hierarchies • to handle complex systems • requires definition of references 	

<p>Martin Schindler Software Engineering RWTH Aachen University Page 13</p>	<h2>4. Concrete Syntax (3/3)</h2> <ul style="list-style-type: none"> ▪ Guideline 20: <i>"Balance compactness and comprehensibility."</i> <ul style="list-style-type: none"> • compact notations enables productivity while writing but can hinder comprehensibility → short notations are more preferable for frequently used elements ▪ Guideline 21: <i>"Use the same style everywhere."</i> <ul style="list-style-type: none"> • improves understandability • eases identification of language elements • user can obtain some kind of intuition for a new language ▪ Guideline 22: <i>"Identify usage conventions."</i> <ul style="list-style-type: none"> • not every aspect should be defined within the language definition (e.g. a certain layout) • conventions describe more detailed regulations that can, but need not be enforced
---	---

<p>Martin Schindler Software Engineering RWTH Aachen University Page 14</p>	<h2>5. Abstract Syntax</h2> <ul style="list-style-type: none"> ▪ Guideline 23: <i>"Align abstract and concrete syntax."</i> <ul style="list-style-type: none"> • eases automated processing, transformations, and presentation (pretty printing) of the model ▪ Guideline 24: <i>"Prefer layout which does not affect translation from concrete to abstract syntax."</i> <ul style="list-style-type: none"> • otherwise using different editors or arranging the model might change its meaning without purpose ▪ Guideline 25: <i>"Enable modularity."</i> <ul style="list-style-type: none"> • enables incremental processing of the models • important for comprehensibility and efficiency in handling large systems ▪ Guideline 26: <i>"Introduce interfaces."</i> <ul style="list-style-type: none"> • to increase flexibility and hiding complexity
---	---

Design Guidelines for Domain Specific Languages

3. Discussion



Discussion

- Depending on language purpose and domain, guidelines might be
 - **contradicting:**
e.g., combining existing languages may introduce conceptual inconsistencies
 - **unimportant:**
e.g., none executable DSLs for documentation cannot introduce inefficient elements
 - **too cost or time intensive:**
e.g., for small DSLs with few users some improvements might not amortize the costs/time



Guidelines have to be matched against **purpose, complexity, and number of users** of the resulting language.

Design Guidelines for Domain Specific Languages



4. Conclusion

Conclusion

- Discussion of **26 guidelines**
 - as a basis for **language design and development**
 - categorized along the development phases
- Guidelines need to be
 - **weighted** and balanced specific to domain and purpose
- Other guidelines are needed for
 - **integrating** DSLs in a software development process,
 - **deploying** it to new users, and
 - **evolving syntax** and existing models in a coherent way