Envisioning Future
Language Workbenches
Where I come from
Precision ≠ Programming

Performance
Scalability
Robustness
Deployment

Formulas, Rules
Data Structures
Tables
Values

+ Tests
All languages shown in this talk are built with the open source JetBrains MPS language workbench.
MPS Language Workbench

Language Workbench
Open Source, by JetBrains
Very Expressive
Used for years in industry
Vast Experience
MPS Language Workbench

- Refactorings, Find Usages, Syntax Coloring, Debugging, ...

Diagram:

- Language
  - Structure: Concepts, Properties, Inheritance, Relationships
    - Provides editors for:
      - Editor: Projection Rules, Side Transformations, Intentions
      - Type System: Typing Rules, Type Checks, Other Validations
      - Transformations: Reduction Rules, Weaving Rules, Transformation Priors
  - Generates to:
    - Language
  - Extends 0..*

- Constraints: Scopes, Usage Restrictions, Property Value Limitations

- Defines execution semantics for:
  - Language
Funktionenmodell berbwekFF

Formale Beschreibung

Funktion: berbwekFF
Programmquelle: vmsctfa1.c
Produkt-Typ: Fonds
PK-Typ: Kapital-Konto
Status: 18.1

Parameter-Attribute

lkm_akt_param
lkm_faell_param
ber_zweck_param
kz_rzw_param

Verwendete VADM-Attribute
Keine verwendeten VADM-Attribute, werden automatisch hinzugefügt

Rückgabeb-Atribut

bwvek

auferufene Funktionen

VTRKermbtgfaelIFF (a)
berbweinzelfF (a, b, c)

Beschreibung

Die Funktion liefert den Barwert per \( \text{lkm}_\text{akt}_\text{param} \) des vorschüssigen Zahlungssstroms \( \text{lkm}_\text{akt}_\text{param} \) bis \( \text{lkm}_\text{faell}_\text{param} \) – jeweils einschließlich. Zahlungszeitpunkt: \( \#\text{lkm}_\text{akt}_\text{param} \) bis \( \#\text{lkm}_\text{faell}_\text{param} \) – 1. Der Parameter \( \text{lkm}_\text{faell}_\text{param} \) steuert die Zahlenweite des Zahlungsstroms. Möglich sind zur Zeit nur die Ausprägungen 0 (Zahl und 12 (monatliche Zahlungsweise).

Hilfsvariablen

kz_bf_hilf

Verarbeitungen

Schleife über \( \text{lkm}_\text{faell}_\text{hilf} = \text{lkm}_\text{akt}_\text{param} \) bis \( \text{lkm}_\text{faell}_\text{param} \)

Falls \( \text{kz}_\text{rzw}_\text{param} = 12 \)

\( \text{kz}_\text{bf}_\text{hilf} = 1 \)

sonst

\( \text{kz}_\text{bf}_\text{hilf} = \text{VTRKermbtgfaelIFF} (\text{lkm}_\text{faell}_\text{hilf}) \)

Ende Falls \( \text{kz}_\text{rzw}_\text{param} = 12 \)

\( \text{bwvek} = \text{bwvek} + \text{kz}_\text{bf}_\text{hilf} \times \text{berbweinzelfF} (\text{lkm}_\text{akt}_\text{param}; \text{lkm}_\text{faell}_\text{hilf} - 1, \text{ber}_\text{zweck}_\text{param}) \)

Ende Schleife

return bwvek
**decision table** BpScoreDecisionTable(sys: bpRange, dia: bpRange) =

<table>
<thead>
<tr>
<th>sys</th>
<th>&lt;= 90</th>
<th>[91..140]</th>
<th>[141..150]</th>
<th>[151..160]</th>
<th>[161..179]</th>
<th>&gt;= 180</th>
</tr>
</thead>
<tbody>
<tr>
<td>dia &lt;= 50</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>[51..90]</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>[91..95]</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>[96..100]</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>[101..109]</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>&gt;= 110</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

**decision tree** DiarrheaStoolsDecisionTree(score: DiarrheaStoolsOverBaseline, patientHasAnySymptom: boolean, goToStartBrat: boolean)

- patientHasAnySymptom
- score >= 7
- score in [4..6]

**type** temperature: number[36..42]{1}
**type** measuredTemp: number[35..43]{2}

val T_measured: measuredTemp = 42.22
val T_calibrated: temperature = T_measured * 0.93

Error: type number[32.55..39.99]{4} is not a subtype of number[36..42]{1}
**Unterhaltsvorschuss**

Zeitangabe: laufend  
Häufigkeit: monatlich einmal  
Leistungskontext:  
Leistungsart: Leer  
Zählart: uvg  
Anspruch Beginn: Anfang - Unbegrenzt: junger Mensch.geburtsdatum

Anspruch Ende: 01.01.1800  - 31.12.9999 : min(junger Mensch.geburtsdatum + 12 Jahre)

Zeitraum für Berechnung: Anfang - Unbegrenzt: **true**
zweckgebundene Leistung: **false**
dem Grunde nach:  
Zeitraumbezogene Daten:
nullwerte Anzeigen: *boolean* = 01.01.1800  - 31.05.2016  - 01.06.2016 - Unbegrenzt: *false*

berechnungsort: *berechnungsorttyp* = 01.01.1800  - 31.12.9999  
Bezugsobjekte: << ... >>
Attribute:
bemerkung : string wird validiert
antragsdatum : Datum

**Nebenberechnungen**

Name: Kindergeld für vollen Monat

(01.01.1800  - 31.12.9999 )
Rechnungsart: wenn: wird geboren mit junger Mensch als person dann vollen sonst: taggenau
Begünstigtenprinzip: **false**
monatswert = Kindergeld 1. Kind
zwischenergebnisse = [<< ... >>]
dergebnis = monatswert
isore rest rule TimingPattern_07_ObjectsAcrossRows_Modified_TandA_897_1Iteration {

description: 1381R3.ARCHD.0609.TimingPatternLanguage_TestScenarioMap.xlsx

parameters: [TimePeriodObjectTypA4]

patterns:

```
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
---|---|---|---|---|---|---|
scenario |  |  |  | < TimePeriodObjectTypA4 > |  |  |
scenario | TimePeriodObjectTypA1 |  |  |  |  |  |
scenario |  |  |  | < TimePeriodSpecifier2::Duration = 24 Hours > |  |  |
scenario |  |  |  | < TimePeriodSpecifier3::Duration = 15 Minutes > |  |  |
scenario |  |  |  |  |  |  |

△ TimeSpikeObjectTypA5
```
risk factor Ausrüstung with levels
- Standard: Gängige IT-Ausstattung (z.B. Notebook, frei verfügbare OBD-Diagnose-Tools) = 0
- Spezialausrüstung: Professionelles Werkstattequipment (CAN-Cards, Diagnoseequipment) = 4
- Maßanfertigung: Mindestens ein Sonderwerkzeug (z.B. nur für Hersteller verfügbares Werkzeug oder Kostenintensive Ausstattung >50.000 € (z.B. Elektronenmikroskop)) = 7
- Mehrfache Maßanfertigungen: Mehrere Sonderwerkzeuge = 9

risk factor Kenntnis des Zielobjektes with levels
- Öffentlich: = 0
- Intern: = 3
- Vertraulich: = 7
- Geheim: = 11

required attack potentials
- Sehr gering: = 0
- Gering: = 10
- Moderat: = 14
- Hoch: = 20
- Mehr als hoch: = 25

<table>
<thead>
<tr>
<th>Attack Potentials Table</th>
<th>Required consecutive attack potential</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mehr als hoch</td>
</tr>
<tr>
<td>Mehr als hoch</td>
<td>Mehr als hoch</td>
</tr>
<tr>
<td>Hoch</td>
<td>-</td>
</tr>
<tr>
<td>Moderat</td>
<td>-</td>
</tr>
<tr>
<td>Gering</td>
<td>-</td>
</tr>
<tr>
<td>Sehr gering</td>
<td>-</td>
</tr>
</tbody>
</table>
public functional component DriveTrain {
    produces SpeedFromEngine
    produces EngineStatus
    produces Gear where it < gearsCount
    consumes RoadConditions
    param int gearsCount
    consumes DrivingCommands
}
vector<
int16,
3>
aVector = \begin{bmatrix} 1 \\
2 \\
3 \\
\end{bmatrix} * 512;

vector<
int16,
3>
resultOfCrossProduct = aVector x aVector;

matrix<
int16,
2x3>
\text{aMatrix} = \begin{bmatrix} 1 + 2 & 2 * 7 & 42 \\
3 & 51 & 24 \\
\end{bmatrix};

matrix<
int16,
3x2>
transposedMatrix = aMatrix^T;

int32 averageIntArray(int32[] arr, int32 size) {
    \text{return} \frac{\sum_{i = 0}^{size} arr[i]}{size};
}

averageIntArray (function)
```plaintext
val beitragsProzentsatzArbeitnehmer: 1.50%  
val beitragsProzentsatzArbeitgeber: 1.50%  

daten ArbeitslosenversicherungStamm {  
  beitragsgruppe : arbeitslosenversicherungBeitragsgruppe  
  unternehmenRechtskreisOst : boolean  
}

ergebnis [monatlich] ArbeitslosenversicherungErgebnis {  
  arbeitgeberBeitrag : €€€  
  arbeitnehmerBeitrag : €€€  
}

fun getSvBruttoGekürzt(rechtskreisOst: boolean, svBrutto: €€€): €€€  
=  

<table>
<thead>
<tr>
<th>rechtskreisOst</th>
<th>svBrutto &gt; bbgOst</th>
<th>svBrutto &gt; bbgWest</th>
<th>wert: €€€</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td></td>
<td>bbgOst</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td></td>
<td>bbgWest</td>
</tr>
<tr>
<td>false</td>
<td></td>
<td>true</td>
<td>svBrutto</td>
</tr>
</tbody>
</table>
```
MPS: Notational Freedom

100% crucial for acceptance in domain!

Rabbit Hole

Modeling is more than diagrams!
MPS: Language Composition

**Embedding**

\[ L_{\text{Host}} + L_{\text{Adapt}} + L_{\text{Emb}} = \]

**Extension**

\[ L_{\text{Base}} + L_{\text{Ext}} = \]

**Extension Composition**

\[ L_{\text{Base}} + L_{\text{Ext1}} + L_{\text{Ext2}} = \]

SPLE on Language Level!
Language Extension Patterns

GPL Extension
Reuse GPL incl. Expressions and TS
Add/Embed DS-extensions
Compatible notational style
Reduce to GPL

New Language
Analyze Domain to find Abstractions
Define suitable, new notations
Rely on existing behavioral paradigm
Reuse standard expression language
Interpret/Generate to one or more GPLs

Formalization
Use existing notation from domain
Clean up and formalize
Generate/Interpret
Often import existing „models“
Realtime Incremental Transformations
Example Use Cases

- Mapping of system models to model checkers for interactive verification of temporal properties
- Flattening of component hierarchies for type checking
- Flattening of hierarchical feature models for path expression evaluation
- Weaving in of safety concerns into C code.
- Desugaring of business DSLs to a core functional language; that core language would then feature an interpreter, a compiler and an integration with an SMT solver
What is Shadow Models?

Functional **transformation language** with support for fixpoints

**Incremental** execution upon change of input model

Unidirectional, but with first-class support for **lifting** results

Fully integrated into **MPS IDE**

**Code:** https://github.com/JetBrains/MPS-extensions/tree/master/code/shadowmodels

**Docs:** https://jetbrains.github.io/MPS-extensions/extensions/shadowmodels/
Mechanics

User edits the input model

A delta is propagated into the transformation engine

A change on the shadow model is produced

This triggers analysis or update of results in Shadow Model

Results/messages are lifted back to input level

Results are annotated to the input model

Might be stacked
Kf2

• Minimal, Expressive Core
  • Functional
  • Reactive
• Sugar on top
• Interpreter, Generator and Verifier below
• Realtime-Trafo with Shadow Models
module TaxExample {

enum states { BW }

  calculation root = sub
    item sub = 10

}

module TaxExample {

  val states_BW = -2084976718

  record root_Env {
    root : int
    sub : int
  }

  fun root_root(values: root_Env) : int = {
    if values.root != 9999
      then values.root
    else root_sub(values)
  }

  fun root_sub(values: root_Env) : int = {
    if values.sub != 9999
      then values.sub
    else 10
  }

}
Semantics
LWBs

Semantics
LWBs + Semantics
Semantics

Functional + Imperative + Reactive

The tools exist: K, Funcons, VDM

But there is very limited integration with LWBs.

Exception: Spoofax + Funcons
Semantics

Grow on (imperative) C

<table>
<thead>
<tr>
<th>User Extensions</th>
<th>User-defined Layer</th>
<th>Platform</th>
<th>MPS</th>
<th>Foundation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Languages shipped with mbeddr</td>
<td>Testing</td>
<td>Importer</td>
<td>Libraries for web server, node navigation, additional notations, pattern matching, palettes, XML processing, debugging...</td>
<td>Syntax Highlighting, Code Completion, Goto Definition, Find Usages, Type Checking, Data Flow Analysis, Refactoring, Versioning, Debugging</td>
</tr>
<tr>
<td></td>
<td>Logging</td>
<td>Concurrency</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Utilities</td>
<td>State Machines</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Messaging</td>
<td>Physical Units</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Components</td>
<td>Component Variability</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Requirements &amp; Tracing</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Documentation</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reports &amp; Assessments</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Visualizations</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>State Machine Verification</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Component Contracts Verification</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C Verification</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Decision Table Checking</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PLE Variability Checking</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Grow on Functional Language...

Several Projects / DSLs

Functional Interpreter

Reactive Interpreter

KernelF
Grow on Functional Language...  
... and other Foundations.

Several Projects / DSLs

Lots of ideas 😊

Functional Interpreter
Reactive Interpreter
Solver
Model Checker
Rules Engine

KernelF
Beyond Balls and Trees

Not too much has happened since the legendary demo.
Overlay program execution data
Run tests and illustrate results in the background.
Instance-based programming (Jonathan Edwards)

That’s it?
What else can we do?
Which LWB-support can be build?
Why do people like Excel?
Beyond Balls and Trees

```kotlin
interpreted<false> module TaxExample {

enum states { BW, BY, MV, BE }
fun isEasternState(s: states) = s == MV || s == BE

calculation incomeTax = alt[
  salary < min => 0 EUR
  otherwise => salary * percentage / 100
]

container item Citizen
  item married : bool
  item salary : money
  item state : states
  item min : money = if married
    then 2000 EUR
    else 1000 EUR
  item percentage =
    if isEasternState(state) then
      married | otherwise
      20 | 40
    otherwise
      10 | 30

assert equals run incomeTax with
  married false salary 5000 EUR state BW
  married true salary 5000 EUR state MV
  married false salary 500 EUR state MV
  married true salary 1500 EUR state BW

}
```
Beyond Balls and Trees

Not too much has happened since the legendary demo.
Overlay program execution data
Run tests and illustrate results in the background.
Instance-based programming (Jonathan Edwards)

That's it?
What else can we do?
Which LWB-support can be build?
Debuggers
Debuggers

Different ones for different language paradigms

- Functional => Tree
- Imperative => Stepping
- Declarative => ?
Debuggers

Different ones for different language paradigms

- Functional => Tree
- Imperative => Stepping
- Declarative => ?

Often the (semantic reverse) of generation
it has to recover DSL semantics for watches, views, breakpoints

Related to simulators and „model animators“

Realtime vs. Post Mortem

No good tool support for building them!
Language Parametrization & Interfaces
Language Composition

Embedding

\[ L_{\text{Host}} + L_{\text{Adapt}} + L_{\text{Emb}} = \]

Extension

\[ L_{\text{Base}} + L_{\text{Ext}} = \]

Extension Composition

\[ L_{\text{Base}} + L_{\text{Ext1}} + L_{\text{Ext2}} = \]
Language Composition

Defining „abstract“ languages with holes
Example: State machines with pluggable expression language

What is the type of those that can be plugged in?
Language Types or Interfaces / Megamodels
Example: How do you describe a „valid“ expression language?
Does this include execution semantics?
Tool Concerns
Applications
Applications vs. Modeling Tool

Novice Users need Guidance!

But not like that!
<table>
<thead>
<tr>
<th>Applications</th>
<th>vs.</th>
<th>Modeling Tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Novice Users need Guidance!</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Applications</td>
<td>vs.</td>
<td>Modeling Tool</td>
</tr>
<tr>
<td>--------------</td>
<td>-----</td>
<td>---------------</td>
</tr>
<tr>
<td><strong>Novice Users need Guidance!</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>What am I working on?</strong></td>
<td><strong>Roles</strong></td>
<td><strong>What am I working on?</strong></td>
</tr>
<tr>
<td>Applications</td>
<td>vs.</td>
<td>Modeling Tool</td>
</tr>
<tr>
<td>--------------</td>
<td>-----</td>
<td>---------------</td>
</tr>
<tr>
<td>Novice Users need Guidance!</td>
<td></td>
<td></td>
</tr>
<tr>
<td>What am I working on?</td>
<td>Roles</td>
<td></td>
</tr>
<tr>
<td>What should I do next?</td>
<td>Tasks, Workflow</td>
<td></td>
</tr>
<tr>
<td>Applications vs. Modeling Tool</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Novice Users need Guidance!</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>What am I working on?</th>
<th>Roles</th>
</tr>
</thead>
<tbody>
<tr>
<td>What should I do next?</td>
<td>Tasks, Workflow</td>
</tr>
<tr>
<td>How complete is my model?</td>
<td>Model States</td>
</tr>
<tr>
<td>Applications</td>
<td>vs.</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>-----</td>
</tr>
<tr>
<td>Novice Users need Guidance!</td>
<td></td>
</tr>
<tr>
<td>What am I working on?</td>
<td>Roles</td>
</tr>
<tr>
<td>What should I do next?</td>
<td>Tasks, Workflow</td>
</tr>
<tr>
<td>How complete is my model?</td>
<td>Model States</td>
</tr>
<tr>
<td>What is the quality of my model?</td>
<td>Metrics</td>
</tr>
</tbody>
</table>
Applications vs. Modeling Tool

Novice Users need Guidance!

What am I working on? Roles
What should I do next? Tasks, Workflow
How complete is my model? Model States
What is the quality of my model? Metrics

Defined jointly with the language.
Permissions
Permissions

Which **user groups** can see and edit which part of the model?

Can we define **views** for showing parts or aspects of models.

**Proxies** for „hidden details“?
Version Control Systems are a problem

**NAME**

`git-pick-remote` — pick the non-reset remote remotes below various diffed staged heads

**SYNOPSIS**

`git-pick-remote [ --diagnose-pack | --split-attract-base | --direct-history ]`

**DESCRIPTION**

`git-pick-remote` picks various staged remotes opposite of various relinked paths, and after a `git-promote-stage` (fetched by `git-observe-subtree` or `git-transfer-tip`) stages a subtree, unsuccessfully quitimported archives are reset for the `git-clarify-stage` command, and upstreams that were stripped during removing are left in a patched state.

`git-edge-base --smack-history` may perform a passive `git-launch-archive` before reverting the object. After checking out paths to many bases, you can fix the head of the changes, as various grepmed areas that were earlier quitimported to the automatic upstreams are filter-branched to a temporary origin. `git-apprehend-area --consult-abduct-head` will apply a passive `git-design-file` before doing anything else.

Provided that `<sharpen-stash>` is not merged, `git-monitor-subtree --swallow-link-history` may apply a temporary `git-relieve-file` before doing anything else, because some bundled trees that were previously ticked over the temporary files are relinked to an automatic pack. In case `DOMINATE_OTHER_FILE` is not checked, `git-define-tip` takes options applicable to the `git-lessen-subtree` command to check what is pruned and how, and `<weight-head>` is format-patched to archive the commit of any heads opposite of the path.

When `git-pick-index` imports an upstream, the user will filter-branch the tips or run `git-slash-history --whack-tip` instead. The `--cultivate-calculate-tip` option can be used to relink a pack for the tag that is checked out by a staged tag, so the `--jar-collate-stash` argument can be used to relink a submodule for the stage that is cloned by a staged origin. If `GRADUATE_REMOTE_PATH` is applied, you can bisect some tips and run `git-dramatize-submodule --support-review-ref instead.`

If `git-crash-submodule` exports a tag, after blaming stashes to many branches, you can remote the pack of the stashes. The same set of commits would in any cases be added in an automatic remote, so `git-stack-submodule` takes flags relevant to the `git-page-log` command to check what is fetched and how.

**OPTIONS**

```
--diagnose-pack
  the tree may not be balanced by a reverted file

--split-attract-base
  with this argument, `git-file-commit --obtain-nail-tag` relinks origins that remote the supplied upstrams

--direct-history
  push the heads of the tips that are returned
```

**SEE ALSO**

`git-install-index(1), git-feed-tag(1)`
Version Control Systems are a problem

https://git-man-page-generator.lokaltog.net/

But file-based version control is often not suitable.
Google Docs Style Collaboration

Realtime collaboration à la Google Docs

Language/structure-aware

Some level of transaction or isolation

Some version of branching.
The cloud
Cloud Backend

Local vs. „real“
Data Protection!
LSP-style frontend.
Browser Frontends
Become more App-Like

UI must be **less** like an IDE.

Things must look **slicker**!

It must be easier to **integrate** with web apps that are developed without the LWB:

Microservice-style modeling „services“

Integration on common backend and also with non-model apps.
Bring in M0

M3
Language DSLs
Part of the LWB

M2
Language
Defined using the LWB
Should feel like IDE

M1
Model
Should be on the web
Not feel like IDE

M0
User Data
Completely outside in generated App/DB
Bring in M0

Part of the LWB

Bring into joint repository.
Simplified Migration of models.
MetaEdit+ does this.
Bring in M0

M3

Language DSLs

Part of the LWB

Bring into joint repository.
Simplified Migration of models.
MetaEdit+ does this.

User-data also needs (incremental) analysis, transformation and migration.
Should also scale indefinitely.
Why „externalize“ through generation etc?
Get Rid of IDE feel
Make it simpler.

In terms of UI

But also in terms of paradigm.

Is functional or imperative the answer?

https://alarmingdevelopment.org/
Make it modular(er).
Explain
Communicate
Convince
Educate
Maybe more important!

How do we call what we do?
End-User programming
Computational Thinking
Case Studies
How do we de-risk tools?
What are the right paradigms?