Outputs from *When Everything Changes: DSL Evolution*

Marina Haase

Warm-up Game:

4 Minutes to answer the question: “What makes DSL refactoring so difficult and error prone”?

- Backwards compatibility (2x)
- Lack of tools
- Migration
- Some tools have too many dependencies
- (re-)train users
- Trade offs
- Prioritization
- Testing
- Lack of goals
- Inconsistent goals
- Unmeasurable goals
- Regenerating
- Tools
- Experts moved on
- Failing editors
- Changing verification rules
- Retest
- Language expressed DSL changes
- Meta-model changes
- Different interpretation of Domain/semantics
- Traceability
- All problems from normal SW-Evolution apply
- Change in syntax and semantics
- Invalidation of legacy code
- Learning the changes (user view)
- Upgrade in tools, backward compatability
- Figuring out what needs to be changes (syn or Sem)
- Testing
- User requirements
- Changes break existing models
- Lacking tool support
- Not planned ahead
- Lacking domain/tool experience

Method: Reverse Brainstorming
Goal → Finding problems in DSL evolution and ideas for solutions

Method Description:
This method uses the fact that people always find it easier to be negative than positive.
Step One: Turn the problem round. What can we do to make the problem worse? What would have to be done so that the problem would definitely NOT be solved.

Imagine you wanted to sabotage a DSL refactoring – what would you do?

Output:
Bold Text: What would you do in order to sabotage a DSL refactoring
Normal Text: Turn the idea around.

Context Sensitivity Intro
- Keep it simple (point of DSLs)
- Think context free alternative

Loose Domain Expertise
- Don't
- Stable management of teams
- Have good document policy
- Make use of apprenticeship

Provide more documentation
- Minimum set
- Summary
- How in contrast to long prose text

Isolate the developer from the users
- Put developer in environment of user
- Clarify use-processes (e.g. reuse required?)

Ignore version 1.0 and start from scratch
- Incremental changes

Deadline was yesterday and you're over budget already
- Apply planning processes, requirements
- keep management aware of costs

Introduce strong typing with AOP
- try to avoid, keep it simple

Let users test (develop without testing) and let developers go on holiday while the users test
- plan for verification,
- define user acceptance test
- define benchmark example

Use Kanjii
- either hire Japanese developers or use ASCII

Require ifdefs to become part of output
- keep it simple

Extend use of macro

- Keep tool domain small, straight

Let source code be accessible to anyone

- Define clear borders
- Define customization interface
- Employ a change of lifecycle management process

Blur the borders of the domain

- Focus on domain

Introduce sub-domain languages

- Wise modularization layers

Require the user to write target code into models

- never

Change target code by hand

- never

Move intelligent analysis into code generator

- intelligence → languages
- backend should be simple and stupid

Multi language output

- follow previous suggestion then no problem

Mess with roles in the DSL

- ditto
- define clear roles

Outsource DSL development to constantly change consultants

- keep it in-house

Method: Structuring Problem Area

Method description

a) Find different ways of dividing DSL evolution into different types of categories – and name the categories.
   e.g. types of transformations
   artefacts that need to be changed
b) Most problems are problems because of forces that pull in different directions (e.g. flexibility vs. simplicity). Solution needs to be a compromise. Try to identify forces involved in refactoring DSLs

Output:

a) Compare photo
b)

- flexibility vs simplicity
- completeness vs noise (for user convenience)
- syntactic sugar vs restricting
- specific vs generic

Rolestorming

Description
Step one:
Collect names of famous people / famous roles you can relate to, or you have some idea of what is/was/would be important to them and write the names on an index card each.

Step two
Go through each index card and brainstorm about the question: "How would "XXX" go about DSL evolution?"

Output:
Obama
"Yes we can" – enthusiasm to change
But does he change very much?
Tries to make everybody happy.
Seeks to be compatible with everyone / past history → is backwards compatible

Al Capone
Do whatever for his own benefit → Disregard backward compatibility
By stealth –code and/or concepts → reinterpret semantics
find a way round the system: e.g. "bribery"

George W. Bush
Not his idea – takes advice from his subordinates
If it works: he will claim the credit: e.g. "My DSL", flag waving
Requirements from sponsorship – in a context with different stakeholders, solution for a particular stakeholder (e.g. the loudest, the best payer, etc.)
Premature release. Stating: "The job is done" – "Mission accomplished" even though it is still incomplete

Stalin
Erase history
Rigid / dogmatic
Short term success but doomed to failure?

Hitler
Change DSL but find a scapegoat for problems of old DSL
Short term success but doomed to failure?

Isaac Newton
Scientific / Analytical approach
Empirical, Experimental – measurements to verify
Proposed theories – then tried to prove them
Grand Unified Approach – most expensive but "beautiful" solution: very flexible but complex

Pope John Paul II
Denial that there’s anything wrong with what there is in the anyway
No evolution

Jesus
Lay down simple principles (not rigorous/dogmatic) for way forward
Band of evangelists to pass on principles – grass roots (e.g. open source)
it will be perfect in the after life (in the next version) → sticking with suboptimal solutions:
It will be all right in the end” – Faith that it will be ok: positive attitude.
He was evolutionary as he built on what was before – but he was also revolutionary
(similar change to C → C++, from legalistic to relational)