through a set of questions and suggests which type of configuration language you need.

2. The *configuration language formulator* allows you to design a configuration language that uses the terms and concepts of your software. This tool strives to find the balance between *easy but not simplistic* and *expressive but not complex*.

Configuration languages are all around us: they support the email rules to better manage your mailbox, decide when to take action in home automation systems and temperature control in regulation systems, etc. Their benefits have been recognized widely.

In Belgium, several companies use them and new pilot cases are starting. Join us!

Contact:

Dr.-Ing. Sebastian Günther
sgunther@vub.ac.be

Dr. Thomas Cleenewerck
tcleenewerck@gmail.com

Vrije Universiteit Brussel
Department of Computer Science
SOFT Research Group
Pleinlaan 2
1050 Brussels
BELGIUM

Website: soft.vub.ac.be/varibru

www.varibru.be

Configuration Languages

Serve more customers with less programming

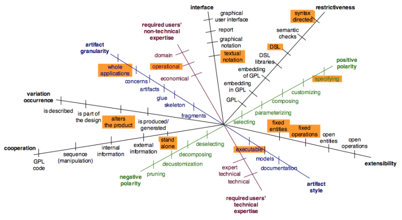
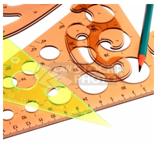
Software provides the opportunity for customer-tailored configuration. However, conventional software engineering techniques lack the necessary power to facilitate the configuration – quite the opposite, more variability often means more pain.

Configuration languages are specifically crafted domain-specific languages built to *hide* the technical difficulties of programming and offering sufficient expressive power to *easily* describe how to customize your product for your customers.

Configuration languages offer several benefits against conventional software engineering:

Conventional Software Engineering	*Configuration Languages*
⊝ *Slow* delivery times	✓ *Faster* delivery times
⊝ Maintenance of existing variations *prevents* the creation of new product variants	✓ *Company growth* by serving more customers
⊝ *High* development costs	✓ *Reduce* the costs of adapting your product
⊝ *Significant risk* of breaking your product by adding new variants	✓ *Strengthen your market position* by anticipating customer demands and *increase your market share* by using new distribution channels

Applying Configuration Languages

	Step	Example Results
1	**Analysis** In the first meeting, we discuss and help you understanding how your variability challenges can be addressed to attain your business goals.	- Business challenge: "How to serve more customers with minimal adaptation to existing software." - Technical opportunity: “It is impossible to maintain dozens of products, but it is feasible to maintain dozens of configurations.”
2	**Design** Using our tools for designing your personal configuration language, we will propose a custom variability solution for your specific needs.	An architectural reference frame and design decisions to guide you.  interface graphical user interface report graphical notation textual notation restrictiveness syntax directed semantic checks DSL DSL libraries embedding of GPL embedding in GPL GPL required users' non-technical expertise domain operational economical artifact granularity whole applications concerns artifacts glue skeleton fragments positive polarity specifying customizing composing parameterizing selecting variation occurrence is described is part of the design alters the product is produced/ generated fixed entities fixed operations open entities open operations extensibility cooperation GPL code sequence (manipulation) internal information external information stand alone deselecting decomposing decustomization pruning negative polarity executable models documentation artifact style expert technical technical required users' technical expertise
3	**Implementation** Implement and deploy your configuration language or tools in a lightweight approach that seamlessly integrates with your development needs.	A configuration language embedded in your deployment method, to facilitate adapting your product or to create new product variations. 

Why to use Configuration Languages?

A scaleable approach to facilitate providing customer-specific functionality

Software companies working on a product start out with a handful of prototypical customers for whom the product was intended. The product is right on target and serves those customers well. In the beginning, as a company grows, dealing with special customer needs is relatively easy. The development team manages to support the differences among these clients very well: some clients got separate product code bases and parameters covered the basic configuration needs. Very soon after the number of clients grow and their requests for customer specific functionality increase, the company realizes that conventional software engineering doesn't scale and that customer specific functionality outgrows simple parameters.

What can they do to solve this?

Configuration languages simplify the software customization. They provide a dedicated language to specify software configuration. Not only are the configurations easy to maintain and extend, but the deployment of a configured program happens fast.

To design, implement, and apply configuration languages, we provide:

1. The *configuration language decider* to reduce the risk of accidently implementing the wrong kind of configuration language. It guides you