SPLE

Comparison of Implementation Techniques

Characteristics of Implementation Techniques

Category	Description
Performance	*Does a SPL developed with this approach have better/worse performance?*
Collaborative Development	*Does this approach ease or complicate distributed development?*
Third-party software	*Can we acquire and integrate software/components from 3rd-party sources?*
Granularity	*To which level can we implement features, e.g., line-, file-, or component-level?*
Proactive Development	*Can we implement a SPL with no legacy code present?*
Reactive Development	*Can we easily develop the SPL reactively?*
Extractive Development	*Can we easily extract and modularize features?*
Embedded Systems	*Does this approch increase resource utilization (cf. Performance)?*
Required Skills	*Can developer easily adopt this approach or must learn a new language?*
Scalability	*Can we apply this approach to large number of features?*

Performance

No Overhead	Runtime Overhead
• Version Control Systems • Build Systems • Preprocessors *Code of unselected features is not present in any product, no additional runtime overhead.*	• Frameworks (black- and white-box) • Components & Services • FOP • AOP *These approaches introduce indirections, such as with glue code (frameworks and components) or inheritance (mixins) and original()-calls (jampack), and pointcuts in AOP.*

Collaborative Development

Not Problematic	Problematic
• Version Control Systems (main purpose) • Frameworks/Components *Extensions can be developed independently, dependencies are expressed via glue code.* • AOP • FOP *Obliviousness principle enables development of extensions independent from base code.*	• Build Systems • Preprocessors *Developers can edit the build scripts or preprocessor directives and interfere with other developers.*

Third-party software

Not Problematic	Problematic
• Frameworks • Components *Features are modularized.*	• Version Control Systems • Build Systems • Preprocessors *Features are not modularized and cannot easily be integrated.* • *FOP* • *AOP* *Reusability only within the product line.*

Granularity

Product-Level	**Coarse-Grained** (e.g., file-level)	**Fine-Grained** (text- or method-level)
• Version Control Systems *Development maintains every product on a separate branch.*	• Build Systems *Only files related to a product are included.*	• Preprocessors • FOP (method-level) • AOP (statement-level)

- Frameworks: Determined by the visibility of methods/hot-spots (see preplanning problem)
- Components: Hard to find optimal component size (preplanning- and library scaling problem)

Proactive Development

Not Problematic	Problematic
• *Frameworks* • *Components* *Features are fully modularized, might require glue code (components).* • FOP • AOP *No extensive pleplanning required.*	• Version Control Systems • Build Systems *Only suitable for small numbers of variants (merge conflict, individual build targets)* • Preprocessors *Poor feature traceability.*

Reactive Development

Not Problematic	Problematic
• Version Control Systems • Build Systems • FOP • AOP *Minimal preplanning required.* • Preprocessors *Minimal preplanning required, fine-grained extensions possible.*	• *Frameworks* • *Components* *Later introduction of extensions is hard due to rigid structure.*

Extractive Development

Not Problematic	Problematic
• *Preprocessors* Fine-grained extensions.	• Version Control Systems • Build Systems • Frameworks • Components *Coarse granularity of extensions compilcate extraction, features are not modularized. Later introduction of extensions is problematic (frameworks/components)* • FOP • AOP *Less suitable since feature code needs to be extracted to FOP code or aspects.*

Required Skills

No/Few new skills required	Advanced skillset required
• Version Control Systems • Build Systems • Preprocessors *Well established tools, whose usage serves a different purpose besides SPL engineering. Good tool support.* • Frameworks (Black-Box) • Components Only a few interfaces/the component model need to be understood.	• Frameworks (White-Box) *Deep understanding of the framework is necessary.* • *FOP* • *AOP* Conceptual extensions to OOP, new paradigms, need to understand composition mechanisms (FOP) or learn a new language (AOP).

Scalability

Not Problematic	Problematic
• Frameworks • Components • FOP • AOP *Features are completely modularized, FOP and AOP have been designed for scalability.*	• Version Control Systems • Build Systems *Only tractable for small numbers of variants, merge-conflicts (VCS) and maintenance of build targets.* • Preprocessors *Poor feature traceability.*