派生開発からプロダクトライン開発への漸次的移行プロセスXDDP4SPL

九州大学大学院システム情報科学研究院情報知能工学部門
九州大学大学院統合新領域学府オートモーティブサイエンス専攻
中西 恒夫
E-mail: tun@f.ait.kyushu-u.ac.jp

本発表の内容は平成24～26年度科研費・基盤研究C（課題番号: 24500042, 課題名: プロダクトライン開発の派生開発からの導入と適用範囲拡大に関する研究）の助成の成果によるものです。

目次

- SPL vs. XDDP
- XDDPからのSPL導入（XDDP4SPL）
- XDDP4SPLにおけるコア資産管理手法
- まとめ

SPL vs. XDDP

背景

今日の組込みシステム開発のほとんどは派生製品の開発。

再利用はQCD改善の要

派生製品開発のための再利用方法論

- Clone & Own：　過去に作ったよく似た製品を持ってきて，追加，変更のあったところに関するコードを改変。
- XDDP (Extreme Derivative Development Process)：　既存製品の追加，変更についてのみ要求，仕様を記述し，最後にコードを改変。
- SPL (Software Product Line)：　計画された製品間で共有するコア資産を構築し，コア資産を定められたとおりに再利用。

SPLの概要

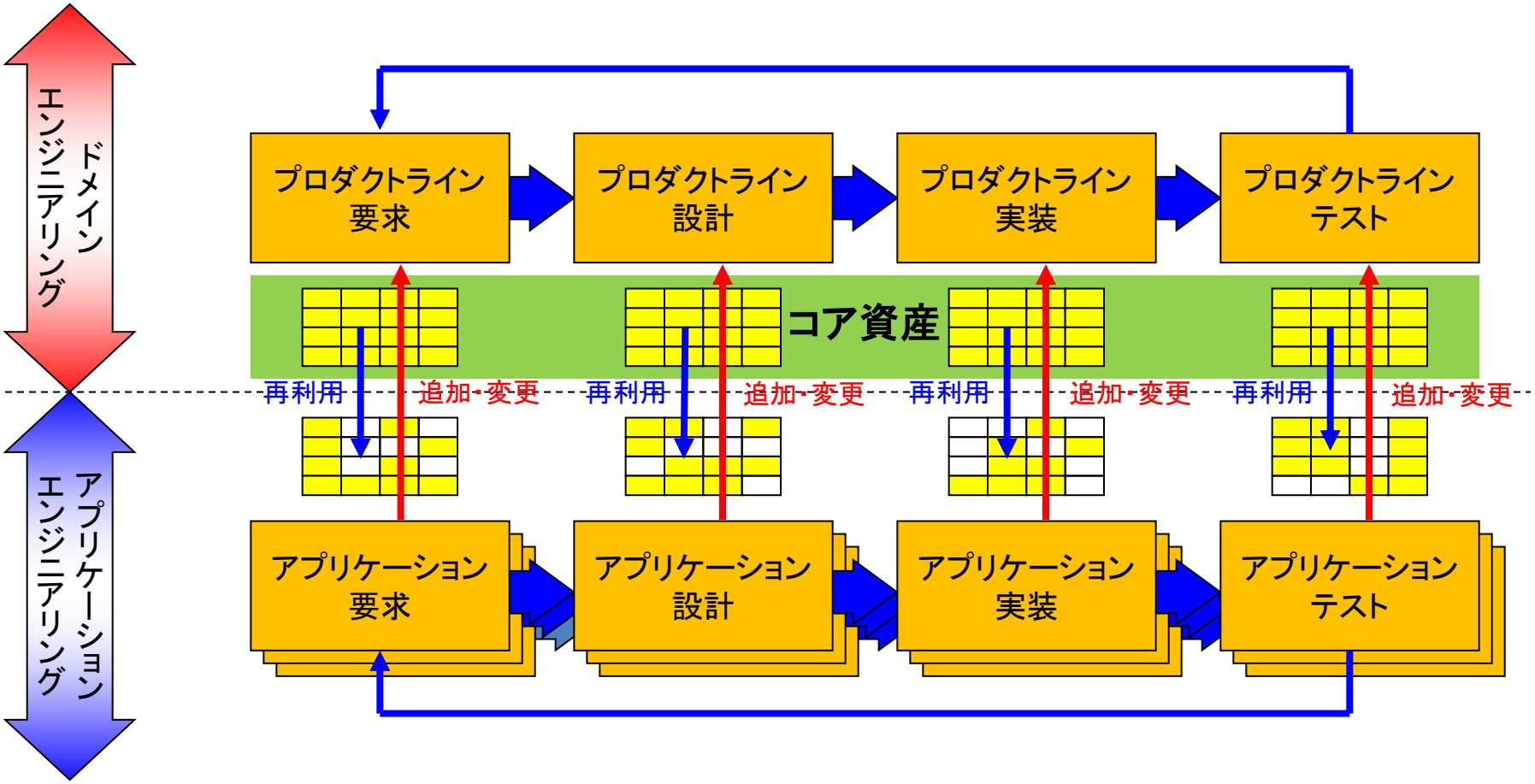
SPLはソフトウェア資産再利用のパラダイム

- まったく未知の製品を開発する再利用技術ではなく，開発の初期段階で将来の製品を可能な限り想定する技術。
- 開発の初期段階でソフトウェア資産の効率的な再利用を計画し，準備する，ソフトウェア再利用技術。
- ボトムアップのコンポーネント指向の再利用技術ではなく，トップダウンのアーキテクチャ指向の再利用技術。
- ソフトウェア資産を再利用するだけではなく，ソフトウェア資産をどのように再利用するかまでを規定。
- 再利用のやり方に縛りをかけることでソフトウェアの再利用性を高め，ソフトウェアの構造的崩壊を防ぐ技術。

SPLの基本概念

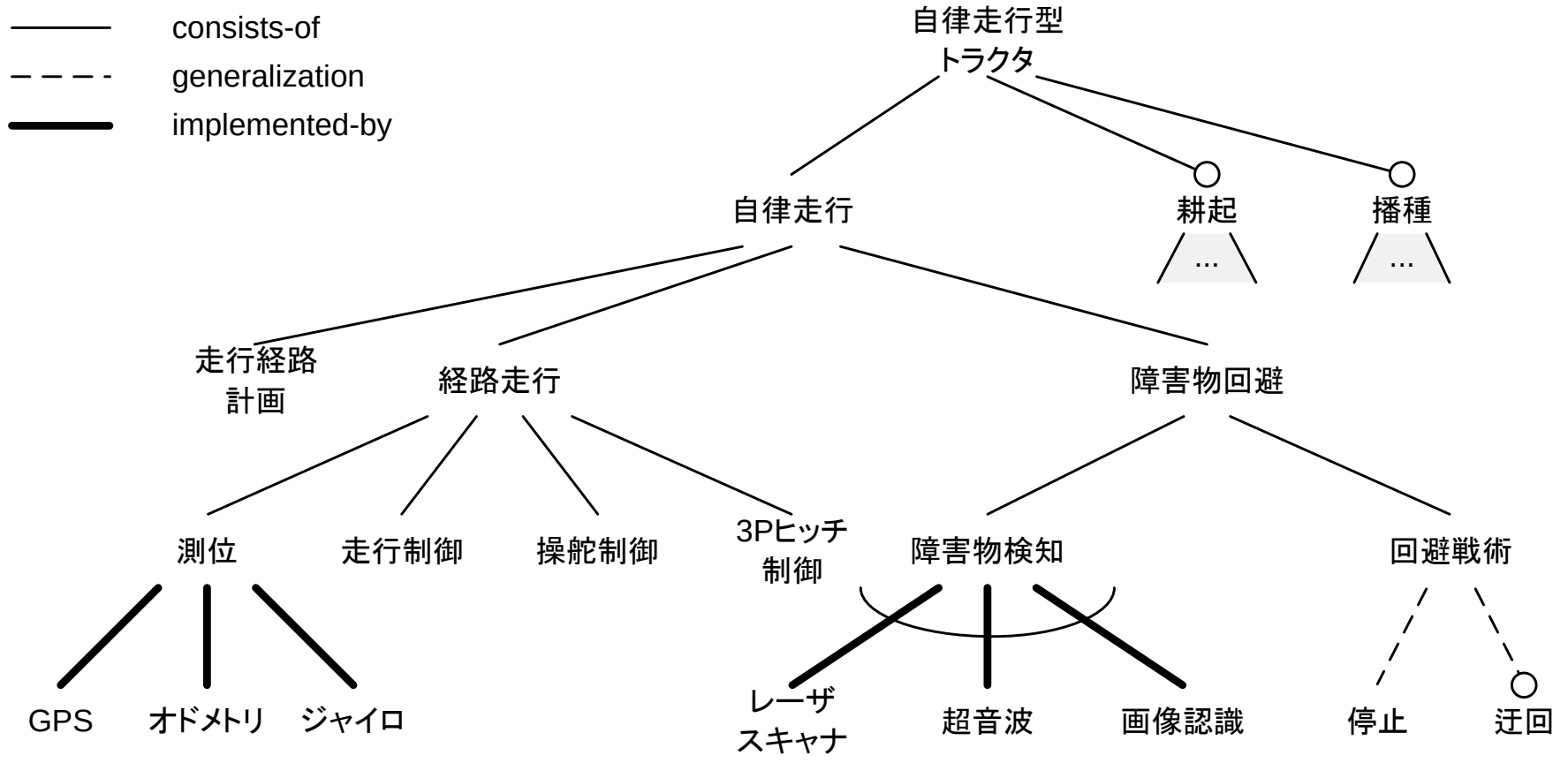
- コア資産
- 二層開発体制： ドメインエンジニアリングとアプリケーションエンジニアリングの分離と並行実施。
- 共通性と相違性の分離
- アーキテクチャ中心開発
- 「ちがい」（問題空間）と「つくり」（解決空間）の分離
- 相違性からコア資産へのトレーサビリティ確保
- 変化点の管理

二層開発体制


ドメイン
エンジニアリング
アプリケーション
エンジニアリング
プロダクトライン
要求
プロダクトライン
設計
プロダクトライン
実装
プロダクトライン
テスト
コア資産
再利用
追加・変更
再利用
追加・変更
再利用
追加・変更
再利用
追加・変更
アプリケーション
要求
アプリケーション
設計
アプリケーション
実装
アプリケーション
テスト

相違性モデリング

フィーチャモデル: 製品の共通点，相違点をフィーチャ単位で示し，かつフィーチャ間の関連を示すモデル。（Kang, 1990/1998）



SPLの導入障壁

SPLの導入障壁はきわめて高い！

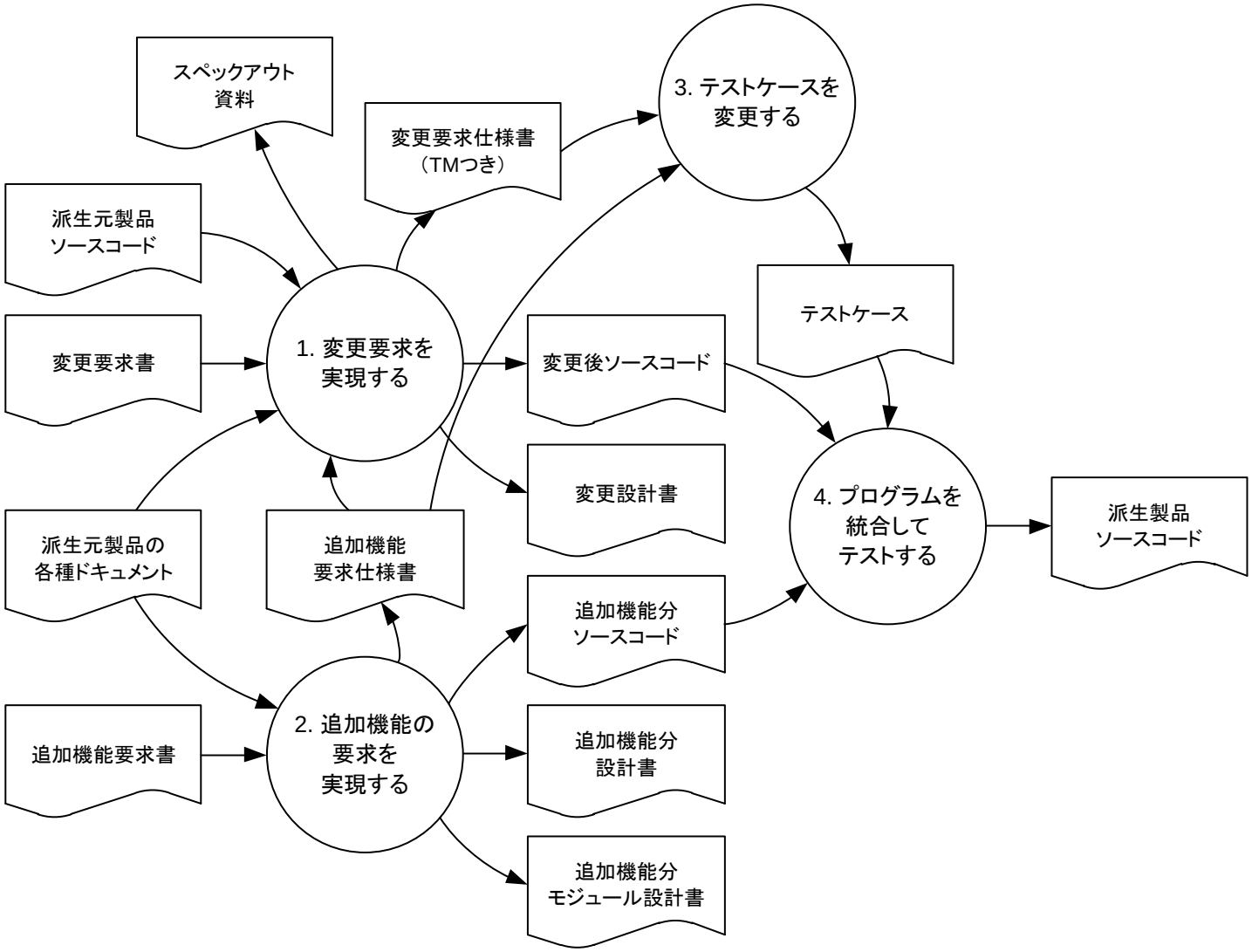
- 将来の製品を予想するのはとても難しい。（特にイノベーティブな製品では！）
- 大規模，複雑なシステムの開発ではドメインエンジニアリングのコストとリスクを受忍しがたい。
- 今日の開発のほとんどは既存システムの強化である。（全く新しいシステムをスクラッチから開発することは稀。）
  - SPL開発ではシステムの全体理解が要求される。
  - システム全体を理解するには十分な量と質の要求，仕様，設計資産が必要である。
  - 開発現場にコードだけしか残っていないことはよくある話。

SPLは漸次的に導入せざるを得ないのが現実。

XDDP: Extreme Derivative Development Process

- 既存製品から新しい製品を派生開発するプロセス。
- 産業界における多くの実践例。
- 既存システムの変更部分に着目。
  - システム全体を理解する必要はない。
  - 追加ならびに変更される機能によって影響を受ける部分のみに既存システムを調査する。
  - 既存コードをどのように追加，変更するか計画を立てる。（コードの追加と変更は最後の最後に行われる。）
- XDDPはSPLではない。
  - コア資産の蓄積は行われない。

XDDPの概要


スペックアウト
資料
変更要求仕様書
（TMつき）
3. テストケースを
変更する
派生元製品
ソースコード
変更要求書
1. 変更要求を
実現する
変更後ソースコード
テストケース
変更設計書
4. プログラムを
統合して
テストする
派生製品
ソースコード
派生元製品の
各種ドキュメント
追加機能
要求仕様書
追加機能分
ソースコード
追加機能要求書
2. 追加機能の
要求を
実現する
追加機能分
設計書
追加機能分
モジュール設計書

SPLとXDDPの比較

SPL	XDDP
パラダイム	メソドロジ
全体理解を志向	部分理解の容認
組み合わせを志向（変化点）	すり合わせの容認
計画駆動	変化駆動
アーキテクチャ志向	アーキテクチャ希薄
変化点の実現に拙速に飛びつかない	コードの修正に拙速に飛びつかない
導入障壁が高い	導入障壁が低い

SPLもXDDPもトレーサビリティを重要視

- SPLはフィーチャからコア資産へのトレーサビリティ
- XDDPは変化／追加要求からコード資産へのトレーサビリティ

XDDPからのSPL導入
（XDDP4SPL）

XDDPからSPLへの移行

理想はSPL

- 基本的にXDDPは間に合わせるための開発プロセス
  - XDDPはコア資産を蓄積しない。
  - XDDPはアーキテクチャの崩れを防いではくれない。
  - XDDPはいつまでも続けるものではない。
- SPLは導入障壁は高いが軌道に乗れば利得大。

XDDP4SPLの思想

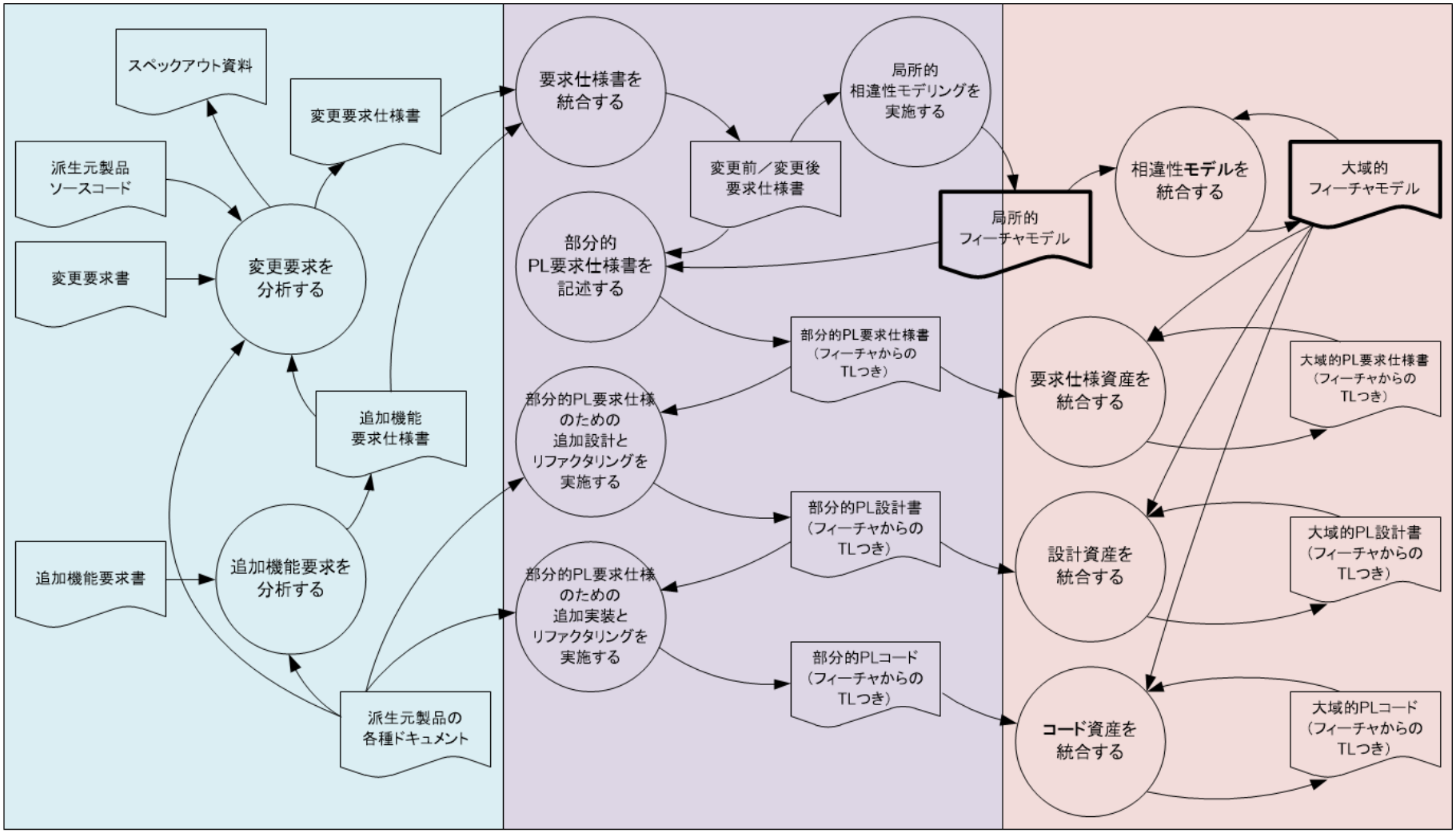
導入障壁の低いXDDPからSPLへの移行を！

- XDDPによる派生開発の中で部分理解を積み重ねて全体理解を進める。
- XDDPによる派生開発のついでにフィーチャとコア資産を発掘，蓄積する。
- 理解の進んだところ，安定しているところから部分的にSPLを実施，徐々に移行する。

XDDP4SPLの概要

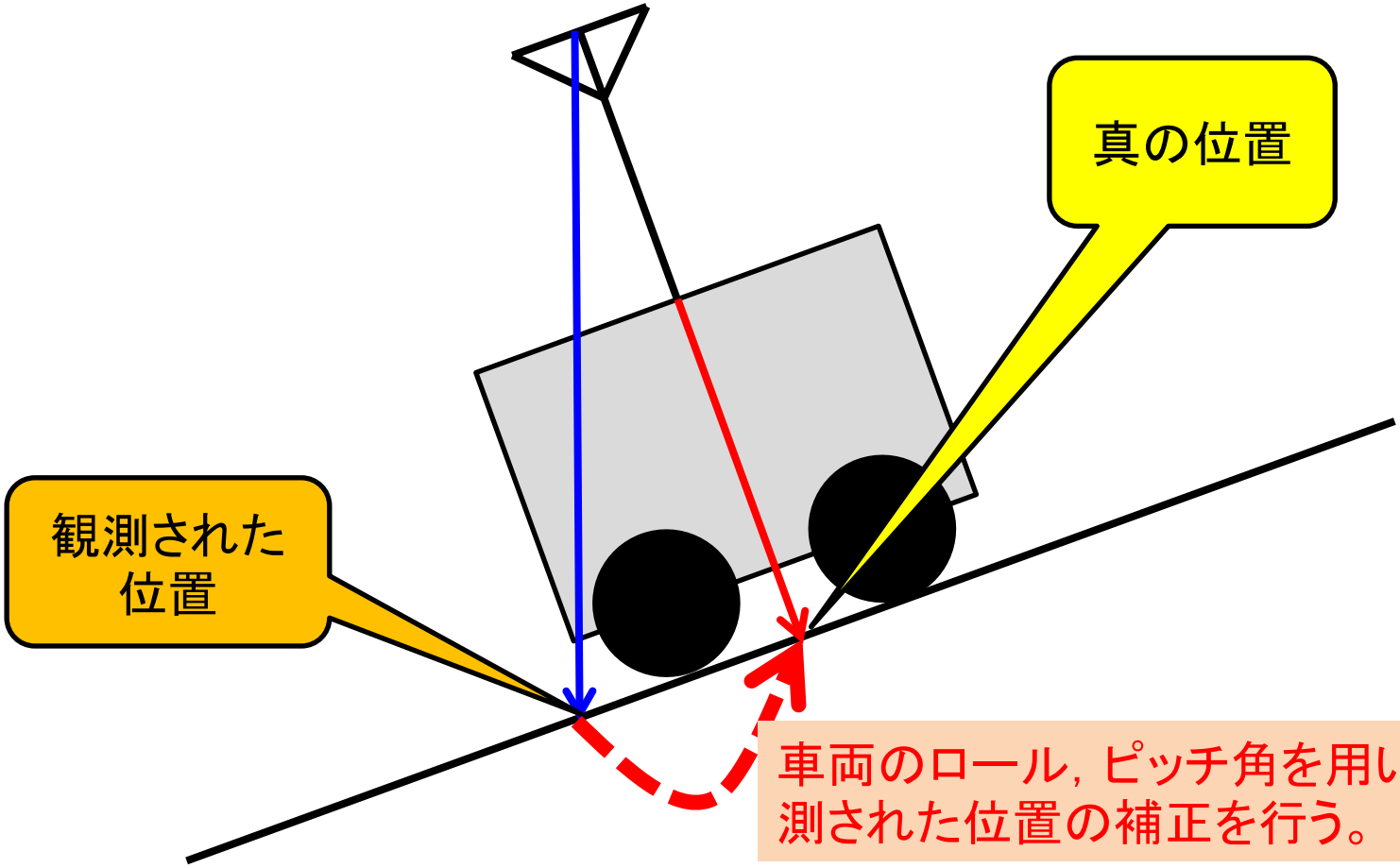
1. 追加機能の要求を分析する。(XDDPと同様)
2. 変更要求を分析する。(XDDPと同様)
3. 1と2を統合し，変更前／変更後要求仕様書を記述する。
4. 3を参照し，局所相違性モデリングを実施する。
5. 3と4を参照し，部分的プロダクトライン要求仕様書を記述する。
6. 5の要求仕様に対して追加の設計，実装，ならびにリファクタリングを実施する。
7. プロダクトラインの部分的なコア資産を統合し，プロダクトライン全体用のコア資産を構築する。

XDDP4SPLの開発プロセス



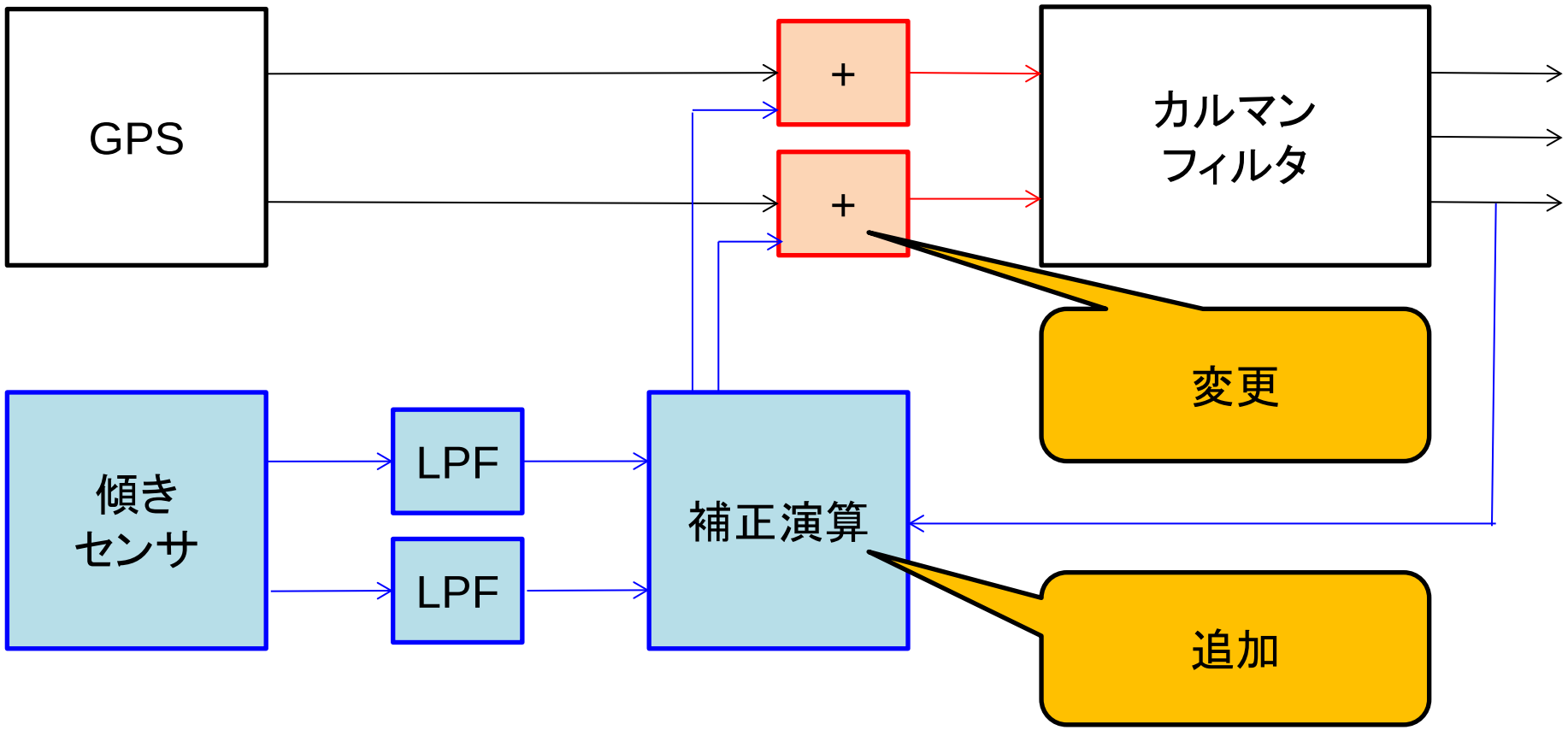
例題：　農業用自律走行ロボット

GPSで観測された位置に対して傾き（ロール＆ピッチ）補正を行う。



変更要求／仕様の記述

GPSで観測された位置に対して傾き（ロール＆ピッチ）補正



変更要求／仕様の記述

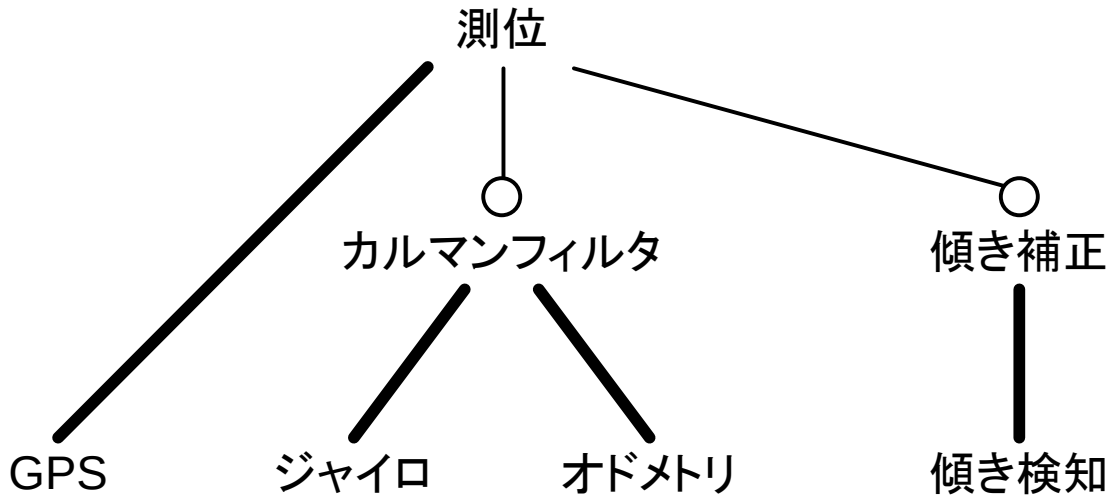
ID	Req/Spec	記述
TiltComp	変更要求	GPSで観測された緯度，経度について，トラクタの傾きによって生じたズレを補正したい。
	理由	GPSはトラクタの作業点から離れた高い位置に設置されるため，トラクタの傾きによって精密な農作業には無視できないほどのズレが生じる。
TiltComp.1	変更仕様	傾きセンサとインターフェースするタスクを追加し，トラクタのロールとピッチを計測する。得られたロールとピッチに対してLPFを適用しノイズを除去する。
TiltComp.2	変更仕様	GPSから得られた生の緯度，経度ではなく，補正した緯度，経度を測位タスク中のカルマンフィルタの入力とする。（カルマンフィルタは測位精度の向上に使われている。）…（略）

変更前／変更後要求仕様書の記述

ID	記述種別	記述
TiltComp	変更前要求	トラクタの傾きを考慮しない，トラクタの現在の位置を知りたい。
TiltComp	変更後要求	トラクタの傾きを考慮した，トラクタの現在の位置を知りたい。
TiltComp.1	変更前仕様	
TiltComp.1	変更後仕様	トラクタのロールとピッチを定期的に計測し，ノイズ除去のために一連の計測値にLPFを適用する。
TiltComp.2	変更前仕様	測位タスクのカルマンフィルタにGPSからの生の緯度と経度を入力する。
TiltComp.2	変更後仕様	測位タスクのカルマンフィルタに傾き補正後の緯度と経度を入力する。…（略）

局所的相違モデリングの実施

システム全体ではなく，追加，変更機能によって影響を受ける部分（とその周辺）についてフィーチャモデリングを実施する。



部分的PL要求仕様書の記述

ID	要求／仕様	記述
GlobalPos	要求	参照点の現在地を知りたい。
GlobalPos.1	仕様	トラクタの現在地，すなわち緯度と経度をGPS受信機から得る。
GlobalPos.2	仕様	傾きセンサからトラクタのロールとピッチを取得し，一連の計測値にLPFを適用してノイズを除去する。[傾き検知]
GlobalPos.3	仕様	GPSから得られた緯度，経度に含まれる，トラクタの傾きに起因するエラーを算出し，GPS受信機から得られた現在地からそれらを減じて，参照点の位置補正を行う。[傾き補正]
GlobalPos.4	仕様	ジャイロセンサからトラクタの向きを取得し，一連の計測値にLPFを適用してノイズを除去する。[ジャイロ]
GlobalPos.5	仕様	オドメトリセンサからトラクタのオドメトリデータを取得し，一連の計測値にLPFを適用してノイズを除去する。[オドメトリ]
…	…	…

アーキテクチャ面の考察（1）

- そもそもソフトウェアの「アーキテクチャ」とは？
- XDDPはアーキテクチャの概念が希薄。
- アーキテクチャが駄目だとSPL的（＝組み合わせ的）再利用の効果が低減。
- SPLは原則的にはアーキテクチャ指向の再利用。
  - SPLの実施レベル（成熟度）は多様である。
  - 成功事例で必ずしもソフトウェアのアーキテクチャが強く認識されている訳ではない。（暗黙的に理解されているケースも少なくない。）
  - プロダクト間でのアーキテクチャの意識，記述，共有ができているSPLはかなりの成熟度。

いつかはアーキテクチャを規定しなければならない。

アーキテクチャ面の考察(2)

アーキテクチャ規定のプロセス(案)

1. as-is のアーキテクチャの理解と記述
   - XDDP/XDDP4SPLにおけるスペックアウト資料の蓄積
   - 局所的フィーチャモデリングの統合
   - リバースモデリング
2. to-be のアーキテクチャの規定
   - 大域的フィーチャモデリングを参照してインターフェースを策定
3. as-is アーキテクチャから to-be アーキテクチャへの移行を XDDP で実施

XDDP4SPLにおけるコア資産管理手法

XDDP4SPLの管理上の課題

XDDP4SPLは管理がカギ

- SPLとXDDPの異なる開発パラダイムの併存
  - 全体理解がされている資産と部分理解のみされている資産の併存
  - コア資産として管理されている資産とそうでない資産の併存
- トレーサビリティ保証とコストのバランス
  - ソフトウェア資産の理解が進まない間はトレーサビリティをとれない。
  - 成長途上の不安定な部分に精密にトレーサビリティをとっても無駄になる可能性が大きい。
  - 最初から精密にトレーサビリティを取ろうとするとドメインエンジニアリングのコストが吊り上がる。

フィーチャ単位でのSPL移行管理（1）

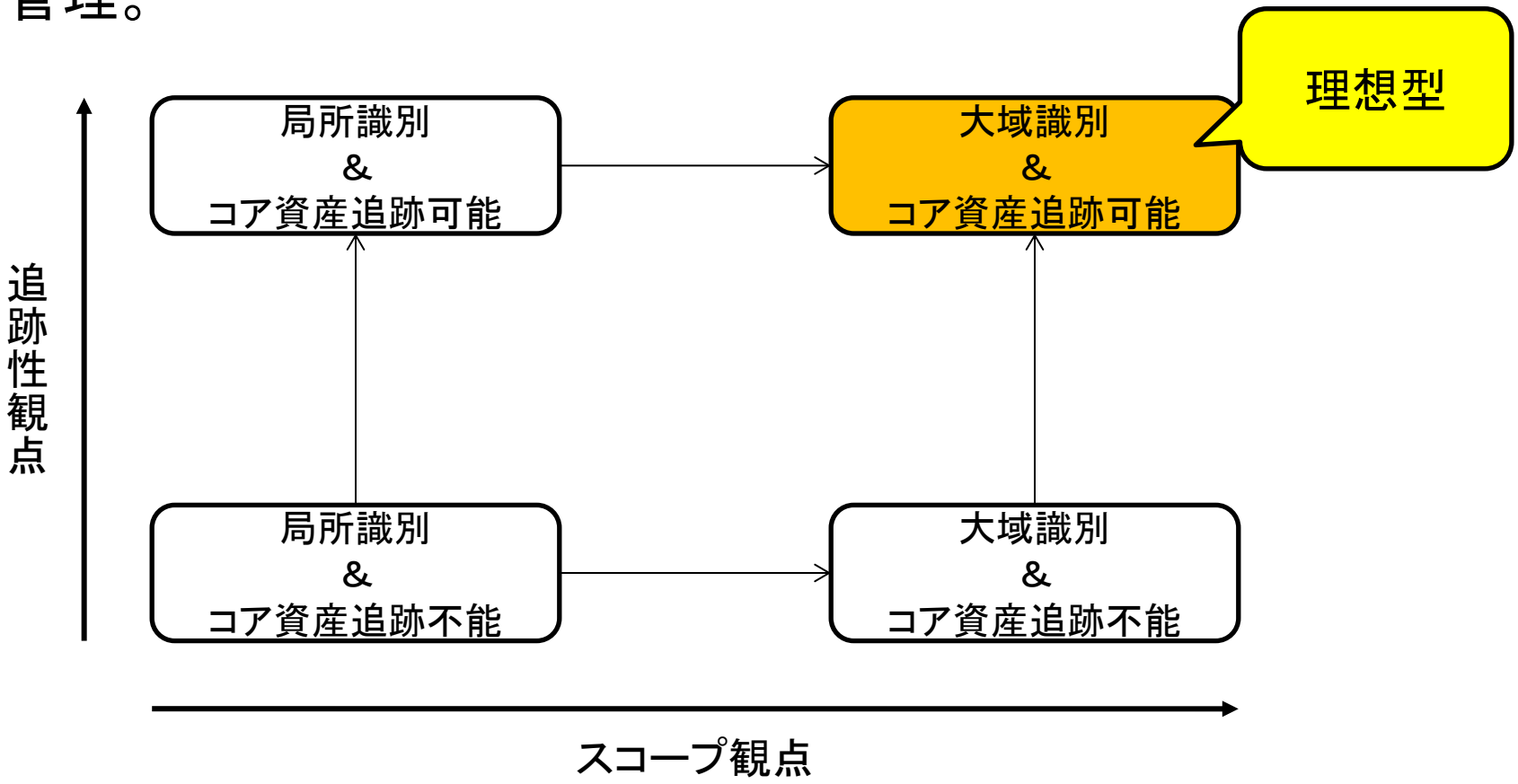
フィーチャの類型

- 局所識別フィーチャ：　プロダクトライン全体の中での位置づけが明らかになっていないフィーチャ
  - コードの局所的なリバースエンジニアリングによって見出されたフィーチャ。
- 大域識別フィーチャ：　プロダクトライン全体の中での位置づけが明らかになっているフィーチャ（＝全体理解がされているフィーチャ）
  - 局所識別フィーチャから昇格したフィーチャ。
  - コードの裏付けはないがプロダクトラインやドメインの知識に基づいてトップダウン的に見出されたフィーチャ。

フィーチャの安定度を評価し，フィーチャ単位でコア資産の整備とSPLへの移行を計画する。

フィーチャ単位でのSPL移行管理(2)

フィーチャ単位のSPL移行管理： フィーチャを局所／大域性ならびにコア資産への追跡可能性の2観点でカテゴリ化し，SPLへの移行を管理。



階層化トレーサビリティマトリックス（1）

階層化トレーサビリティマトリックスの例

ファイル	gps.c						gyro.c						odo.c							loc.c						
関数 大域変数 フィーチャ		gps_init ()	gps_get ()	gps_longitude	gps_latitude	gps_utc		gyro_init ()	gyro_get ()	gyro_roll	gyro_pitch	gyro_yaw		odo_init ()	odo_get ()	odo_rot_fr_l	odo_rot_fr_r	odo_rot_rr_l	odo_rot_rr_r		loc_init ()	getpos ()	kalman ()	tilt_correction ()	longi	lat
測位	+	++	++	++	++								+	?	?	?	?	?	?	+	++	++	++		++	++
GPS	++	++	++	++	++	++																				
カルマン							+	?	?	?	?	?	+	?	?	?	?	?	?	+	?	+	++	?	?	?
ジャイロ							++	++	++	++	++	++														
オドメトリ													++	++	++	++	++	++	++							

ソフトウェア資産の階層構造を反映。できるところまでのトレーサビリティ確保を許す。

階層化トレーサビリティマトリックス（2）

ソフトウェア資産の階層構造

- SPLにおいてコア資産として管理されるソフトウェア資産の抽象度と形式は多様。
- 階層化トレーサビリティマトリックスの階層構造はソフトウェア資産の形式にあわせて規定。

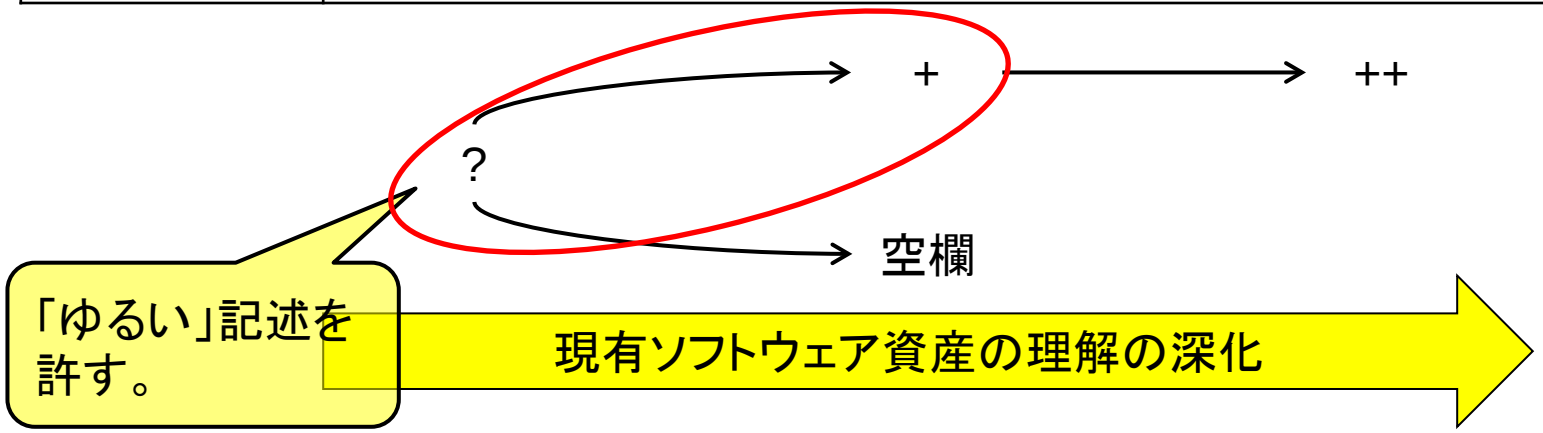
【例】C言語のソースコードの場合：

抽象度／粒度	ソフトウェア資産要素
1（高）	ファイル（.c／.h）
2	関数，マクロ，型（構造体，共用体，列挙型を含む） *etc.*
3（低）	関数内のブロック，構造体／共用体／列挙型のメンバ *etc.*

階層化トレーサビリティマトリックス（3）

階層化トレーサビリティマトリックスの記法

記号	意味
?	フィーチャの実現に資産要素が寄与するかどうか不明である。
空欄	フィーチャの実現に資産要素が寄与しないことが確定している。
+	フィーチャの実現に資産要素の一部が寄与することが確定している。
++	フィーチャの実現に資産要素の全体が寄与することが確定している。



階層化トレーサビリティマトリックス（４）

階層化トレーサビリティマトリックス作成の戦略

- 最初はフィーチャから粗いソフトウェア資産へのリンクを粗く設ける。（たとえばファイル単位）
- XDDPにおける追加，変更箇所周辺のスペックアウト時にリンクを設ける。
- よく理解できている個所，インターフェースの固まった枯れた個所へのリンクを徐々に細かく設ける。（たとえば関数単位）

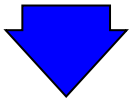
まず緩く！　明らかになったところ，安定したところから硬く！

階層化トレーサビリティマトリックスの保守（1）

理想： 一度定義したフィーチャやフィーチャモデルは以後一切変更されない。

現実： フィーチャもフィーチャモデルも変化する。

- プロダクトラインの成長
- 派生製品の開発
- 現有ソフトウェア資産に関する理解の深化
- プロダクトラインやドメインに関する知識の獲得

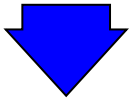


フィーチャやフィーチャモデルの変化により階層化トレーサビリティマトリックスに変更が生じる。

階層化トレーサビリティマトリックスの保守（2）

フィーチャの原始的操作として以下を考える。

- フィーチャの定義
- フィーチャの削除
- フィーチャの名称変更
- フィーチャの意味追加
- フィーチャの意味削減



それぞれの原始的操作によって生じる階層化トレーサビリティマトリックスへの変更を手順化。

階層化トレーサビリティマトリックスの保守（3）

フィーチャの定義

1. 当該フィーチャに対応する行を追加。
2. 同行のすべてのセルに「?」を記入。

ファイル	gps.c						gyro.c						odo.c							loc.c						
関数 大域変数 フィーチャ		gps_init ()	gps_get ()	gps_longitude	gps_latitude	gps_utc		gyro_init ()	gyro_get ()	gyro_roll	gyro_pitch	gyro_yaw		odo_init ()	odo_get ()	odo_rot_fr_l	odo_rot_fr_r	odo_rot_rr_l	odo_rot_rr_r		loc_init ()	getpos ()	kalman ()	tilt_correction ()	longi	lat
GPS	++	++	++	++	++	++																				
GPS測位	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
GPS計時	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

階層化トレーサビリティマトリックスの保守（4）

フィーチャの削除：　当該フィーチャに対応する行を削除。

ファイル	gps.c						gyro.c						odo.c							loc.c						
関数 大域変数 フィーチャ		gps_init ()	gps_get ()	gps_longitude	gps_latitude	gps_utc		gyro_init ()	gyro_get ()	gyro_roll	gyro_pitch	gyro_yaw		odo_init ()	odo_get ()	odo_rot_fr_l	odo_rot_fr_r	odo_rot_rr_l	odo_rot_rr_r		loc_init ()	getpos ()	kalman ()	tilt_correction ()	longi	lat
GPS	++	++	++	++	++	++																				
~~GPS測位~~	+	++	++	++	++																					
~~GPS計時~~	+	++				++																				

階層化トレーサビリティマトリックスの保守（5）

フィーチャの名称変更： フィーチャの意味に変化は生じないため，階層化トレーサビリティマトリックスに変更の必要は生じない。

ファイル	gps.c						gyro.c						odo.c							loc.c						
関数 大域変数 フィーチャ	gps_init ()	gps_get ()	gps_longitude	gps_latitude	gps_utc		gyro_init ()	gyro_get ()	gyro_roll	gyro_pitch	gyro_yaw		odo_init ()	odo_get ()	odo_rot_fr_l	odo_rot_fr_r	odo_rot_rr_l	odo_rot_rr_r		loc_init ()	getpos ()	kalman ()	tilt_correction ()	longi	lat	
2D測位	+	++	++	++	++								+	?	?	?	?	?	?	+	+	+	+			
GPS	++	++	++	++	++	++																				
…																										

「測位」から変更

階層化トレーサビリティマトリックスの保守（6）

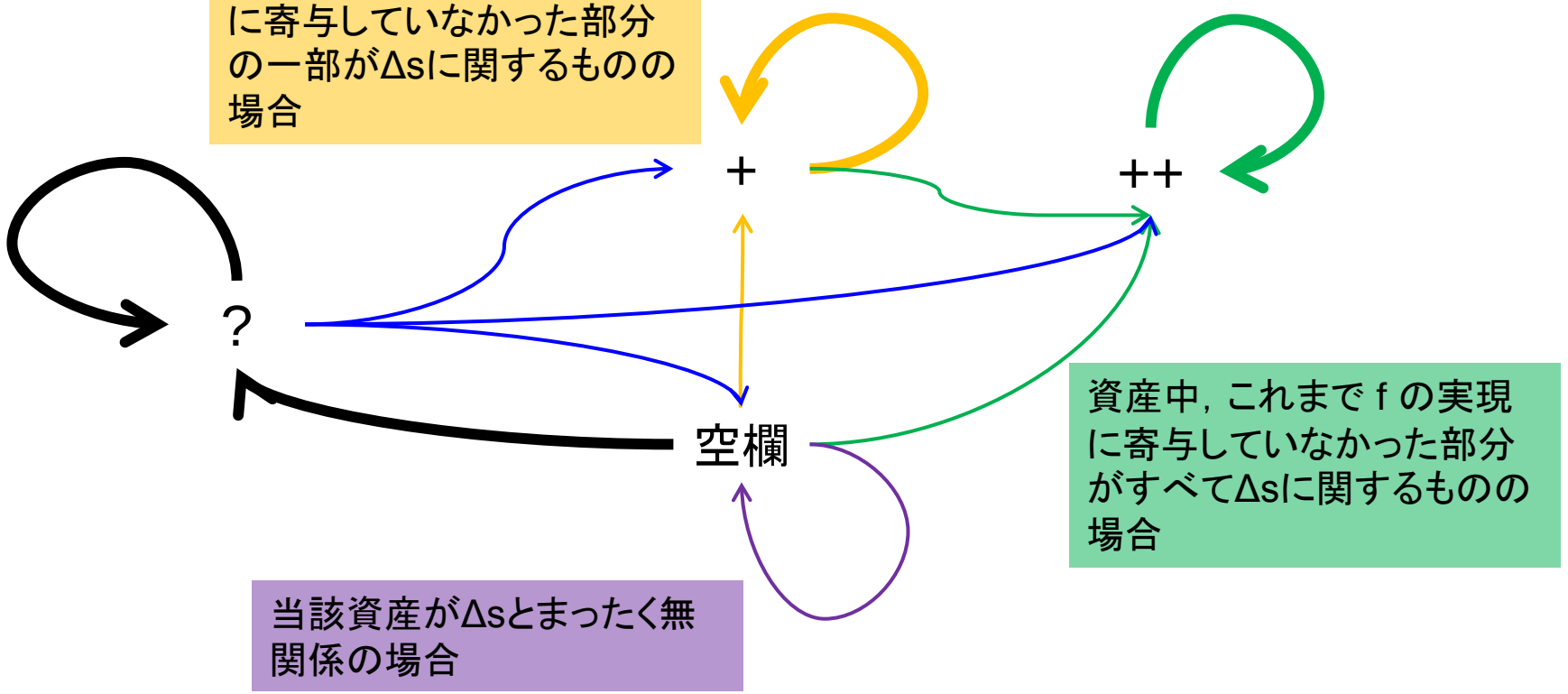
フィーチャの意味追加（+Δs）：　各列の資産において，これまでフィーチャの実現に寄与していなかった部分に影響が出る。

ファイル	gps.c						gyro.c						odo.c							loc.c						
関数 大域変数 フィーチャ		gps_init ()	gps_get ()	gps_longitude	gps_latitude	gps_utc		gyro_init ()	gyro_get ()	gyro_roll	gyro_pitch	gyro_yaw		odo_init ()	odo_get ()	odo_rot_fr_l	odo_rot_fr_r	odo_rot_rr_l	odo_rot_rr_r		loc_init ()	getpos ()	kalman ()	tilt_correction ()	longi	lat
測位	+	++	++	++	++								+	?	?	?	?	?	?	+	++	++	+		++	++
傾き補正																				+	+	+		++	++	++
…																										

「Δs = 車両の傾きを考慮に入れる」を追加。

階層化トレーサビリティマトリックスの保守（7）

フィーチャの意味追加： フィーチャ f の現在の定義に対して，意味Δsを追加する場合，当該フィーチャに関する行の記号を以下のように変更。



階層化トレーサビリティマトリックスの保守（8）

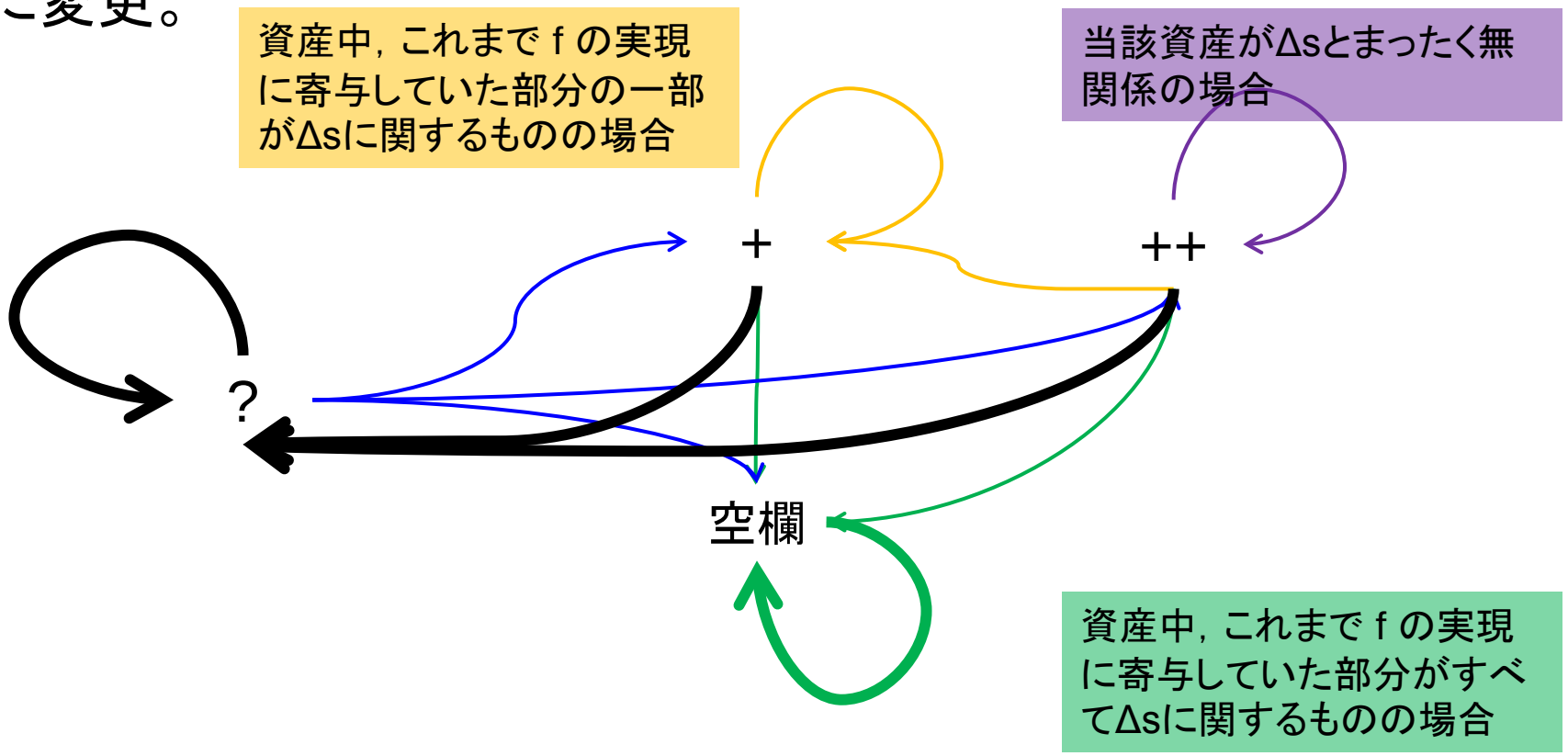
フィーチャの意味追加（–Δs）： 各列の資産において，これまでフィーチャの実現に寄与していた部分に影響が出る。

ファイル	gps.c						gyro.c						odo.c							loc.c						
関数 大域変数 フィーチャ		gps_init ()	gps_get ()	gps_longitude	gps_latitude	gps_utc		gyro_init ()	gyro_get ()	gyro_roll	gyro_pitch	gyro_yaw		odo_init ()	odo_get ()	odo_rot_fr_l	odo_rot_fr_r	odo_rot_rr_l	odo_rot_rr_r		loc_init ()	getpos ()	kalman ()	tilt_correction ()	longi	lat
測位	+	++	++	++	++								+	?	?	?	?	?	?	+	++	++	+	++	++	++
~~カルマン~~							+	?	?	?	?	?	+	?	?	?	?	?	?	+	?	+	++	?	?	?
…																										

「Δs = カルマンフィルタによるデータ統合」を削除。

階層化トレーサビリティマトリックスの保守（9）

フィーチャの意味削減： フィーチャ f の現在の定義に対して，意味Δsを削減する場合，当該フィーチャに関する行の記号を以下のように変更。



まとめ

まとめ

- 派生開発からSPLへの漸次的移行を促進するプロセス XDDP4SPL
  - 部分理解から全体理解への計画的移行
  - フィーチャ単位でのSPLへの移行管理
- XDDP4SPLにおけるコア資産管理手法
  - 階層化トレーサビリティマトリックスによる適度な粒度でのトレーサビリティ管理と保守
- フィーチャ，フィーチャモデルの変化に追従する階層化トレーサビリティマトリックスの更新手法を提案。

たぶんパネルで議論になりそうな話題

- as-is のアーキテクチャ理解は？
- XDDPでも重い？
- SPL完全移行は本当に賢い選択か？
- SPL適用が難しい状況は？

SPLC 2014 論文募集

- SPL分野の歴史的，かつ最も権威のある国際会議
- 2014の開催地はフローレンス
- 概要締切 2014年4月4日，原稿締切 2014年4月11日
- 詳細は: http://www.splc2014.net/