Vol.2016-ARC-219 No.36
Vol.2016-SLDM-175 No.36
Vol.2016-EMB-40 No.36
2016/3/25

DevOpsのための動的ソフトウェアプロダクトラインと派生開発

中西 恒夫[1,a)]　久住 憲嗣[2]　福田 晃[2]

概要：動的ソフトウェアプロダクトライン（動的 SPL）は予見されたシステム運用時の変更要求にシステムの動的再構成で対処する技術である。一方，DevOps は運用時に生じた意図しないシステム変更要求に開発および運用プロセスとそれを支えるシステムとで短期間で対処するパラダイムである。本稿では，動的 SPL を DevOps の一実現手段として捉え，システムの開発と運用に係る不確実性の分析に基づいて，動的適応システムとその運用プロセスを構築し，またシステムの運用時に生じたシステムの動的再構成で対処できない要求については，動的適応システムとその運用プロセスの派生開発で対処するプロセス，DSPL4DevOps を提唱する。DSPL4DevOps はコンポーネントとリソースのリポジトリ，ソフトウェア動的再構成機構，リポジトリ更新機構，運用モニタリング機構を備えた動的適応システムの構築を前提とする。

Evolutional Dynamic Software Product Lines and Derivative Development for DevOps

Tsuneo Nakanishi[1,a)]　Kenji Hisazumi[2]　Akira Fukuda[2]

Abstract: Dynamic software product lines (or DSPL) is a set of technologies to realize predicted requirements calimed during system operation by dynamic reconfiguration of the system. On the other hand, DevOps is a paradigm to manage unpredicted requirements claimed during system operation by development and operation processes and the system supporting the process. This paper deals DSPL as one scheme realizing DevOps and proposes a DSPL4DevOps process. The process constructs a dynamic adaptive system and its operation process based on the result of uncertainty analysis and, for requirements that cannot be resolved by dynamic configuration, performs iteration of derivative/evolutional development of the dynamic adaptive system and its operation process to realize them. The process assumes construction of the dynamic adaptive system to equip component and resource repositories, software dynamic configuraion mechanism, repository update mechanism, and dynamic monitoring mechanism.

1. はじめに

ソフトウェアプロダクトライン（SPL: Software Product Line）[1] は，多分の共通部を持ちつつも，市場のニーズによって生じた機能面，非機能面でのちがいを備えた幾多もの製品を，コア資産の再利用によって，品質，コスト，工期を保ちつつ開発するパラダイムである。SPL のパラダイムに基づく実製品の開発事例はこれまで国内外の産業現場で数多く報告されており，開発現場への導入の問題さえクリアできれば，品質，コスト，工期の改善に大きく寄与し得る手法として認知されているのが現状であろう。SPL の導入にあたって，開発の組織やプロセス，予算等のマネジメント等の非技術的な問題は，依然，SPL コミュニティが一般解を提示できず，成功例の報告等，事例ベースの分析に留まっていることは否定できない。技術的な面では，十分な資料が残っていない，既存資産の全容理解（あるいはアーキテクチャの回復）に係る手間と時間は依然大きな問題であるが，SPL のパラダイムや各種方法論，技法はおお

[1] 福岡大学
Fukuoka University, Fukuoka, 814–0180, Japan
[2] 九州大学
Kyushu University
[a)] tun@fukuoka-u.ac.jp

よそ完成していると言っても過言ではないであろう。

旧来の SPL は市場の多様なニーズやその変化に追随して異なる製品をタイムリに提供していこうとするアプローチである。しかしながら，システムに対する顧客の要求は，ビジネスあるいは技術的環境の変化によって，システムを運用している間にも変化していく。長期間，継続的に運用することが望まれるシステムの場合，変化のたびにシステムを更新していくことは顧客にとって負担が大きく，システムを運用しつつも更新していくことが望ましい。このように運用時にシステムを変化させることを可能とすべく，SPL コミュニティでは動的 SPL（Dynamic SPL）[2], [3] と呼ばれるコンセプトが提唱されるようになった。動的 SPL は必ずしも旧来の SPL のように異なる製品を開発していく技術ではない。動的 SPL は旧来の SPL の諸概念を継承しているものの，ひとつのシステムをその運用形態にあわせて再構成していくことを目的とする技術と捉えるべきである。

旧来の SPL は再利用計画論であり，将来リリースする製品像を描いたうえで，アーキテクチャを定め，コンポーネントをコア資産として開発あるいは発掘，獲得，蓄積していく方法論である。また，動的 SPL は環境の変化にあわせて運用時にシステムの構成を変更していくことを可能とする適応化計画論である。市場のニーズがあまりに多様であったり変化が早かったり予測ができなかたり，思いもかけない使われ方をしたり，競争的な市場で競合他社の動きが読めなかったりといった理由から，製品群の成長やシステムの運用形態の変化に一定のベクトルを見いだせない場合，旧来の SPL や動的 SPL の技術体系をそのまま適用することは困難を極める。

ウォータフロー開発プロセスでは要求工程において，またスパイラル開発プロセスでは製品出荷前の繰り返しプロセスの中において，プロトタイピングやシミュレーション等の手段をも用いて製品に対する要求を獲得しきる努力を尽くすわけだが，ネットワーク接続され他のサービスや製品と組み合わせて運用されるような製品や多様な物理的環境下で運用される製品については，そのあまりに多様な運用の様態ゆえに，将来にわたる要求を開発段階で獲得しきることは不可能である。そこで運用段階であがってくる要求を開発にフィードバックし，運用と並行して短いリードアイムでシステム改良を追随させる DevOps[4] のパラダイムが注目されるに至っている。概念の発展の経緯こそ異なるものの DevOps と動的 SPL の目指すところは重複している。

本稿では，動的 SPL を DevOps を部分的に実現する手段と捉え，動的 SPL の前段階として開発と運用に係る不確実性分析 [5] を実施し，予想される不確実性に備えて動的適応システムとその運用プロセスを構築するプロセス，ならびに予想されていなかった不確実性に対処すべく動的適応システムとその運用プロセスを成長的に派生開発するプロセスについて論じる。

2. 動的 SPL と DevOps

本節では，本稿で論じる動的 SPL および DevOps の概略と両者の関連について述べ，本稿において両技術をどう捉えるのかを総括する。

2.1 動的 SPL

動的 SPL は，旧来の SPL の概念を拡張し，運用環境にあわせたシステムの動的適応を可能とする技術体系である。旧来の SPL の重要概念として，可変性管理，アーキテクチャ指向開発，スコーピング，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並行実施が挙げられるが，動的 SPL においてはこれらの概念は以下のように再定義される [3]。

- **可変性管理**：運用時にシステムを動的適応させるためのコンフィギュレーションを管理する。
- **アーキテクチャ指向開発**：動的適応機構を備えた単一システムのアーキテクチャを定義する。
- **スコーピング**：動的適応の範囲を定める。
- **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並行実施**：ドメインエンジニアリングは動的適応システムの開発に相当し，アプリケーションエンジニアリングにおいては動的適応機構の運用に相当する。

このように動的 SPL は旧来の SPL を出自とし，その重要な概念を旧来の SPL より仮借し，その目的を旧来の SPL と共通にするところはあるものの，その技術的実践の態様は旧来の SPL とは大きく異なる。

2.2 DevOps

DevOps は，開発と運用のギャップを埋め，バグの修正から機能や構成の追加／変更まで運用であがる，あるいは運用してはじめて明らかになるシステムに対する諸要求を，短期間で実現さしめるパラダイムである。DevOps は，昨今流行する種々の技術コンセプトの例に漏れず，巷間で確たる定義が共有されていないバズワードとも言える語であるが，本稿では文献 [4] の定義を採用するものとする。この定義は技術的手段ではなく技術的目的に基づく定義である。

> DevOps は，システムの変更を企図してからその変更が定常運用されるまでの期間を，品質を落とすことなく縮小しようとする技術的実践の総称である。

この定義に述べられた技術的目的を果たそうとする，プロセス，組織構成，技術，マネジメントの組み合わせは DevOps と呼んでよい。文献 [4] によれば，以下のような

技術的実践を DevOps の典型例としている。

- システム運用を要求の第一の源泉とする。
- システム開発でインシデントに対応する。
- 開発と運用とでシステム展開プロセスを密に共有する。
- システムを継続的に展開する。
- アプリケーションと併せてシステム展開のための基盤を準備する。

2.3 動的 SPL と DevOps の関連

発祥の異なる動的 SPL と DevOps であるが，システムの運用における不確実性に対処するという目的，その目的のために用いられる技術的解決手段については共通点が多く見出される。

運用時の不確実性について，システムの動的適応の技術である動的 SPL では，開発段階の可変性分析の段階において把握され記述されていることを前提とする。一方，運用時にあがるシステムの改変要求の実現の最小化を図る DevOps では，開発段階での不確実性の把握は必須ではない。

アーキテクチャについては，動的 SPL ではシステムの動的適応を実現する機構を備えたアーキテクチャの存在を前提とする一方，DevOps ではアーキテクチャの定義はされたほうがもちろんよいものの必須ではない。

動的 SPL の既存研究は，システムの動的適応を実現する具体的な技術的手段について論ずるもの，あるいはそのケーススタディについては多く存在する（たとえば文献 [6], [7]）。しかし，動的 SPL の成長，運用から開発へのフィードバックに関する研究はあまりなされていない。いわゆる DevOps がこれらの問題に対する解を提示できている訳ではないが，概念的には DevOps のほうが包括的であり，動的 SPL は DevOps を部分的にも実現するひとつの可能な技術解のセットとして捉えることができよう。

3. DevOps のための動的 SPL プロセス

本節では，DevOps のための動的 SPL プロセス，DSPL4DevOps を提案する。DSPL4DevOps では，動的 SPL を DevOps を部分的に実現する技術解として捉え，動的 SPL の諸概念をプロセスに組み込んでいる。

3.1 プロセスの概要

DevOps のための動的 SPL プロセス「DSPL4DevOps」は大きく初回プロセスと改善プロセスとで構成される。初回プロセスは動的適応システムとその運用プロセスを実現し，運用し，運用からの要求抽出を行うプロセスであり，最初に一回だけ実行されるプロセスである。改善プロセスは動的適応システムの運用を開始して以降，運用から得られたシステムと運用に関する改善要求に基づいて，動的システムを成長的に実現し，またその運用プロセスを改善し，引き続きこれらを運用し，運用からの要求抽出を続けるプロセスであり，繰り返し実行されるプロセスである。

DSPL4DevOps の初回プロセスを図 1，改善プロセスを図 2 に示す。

3.2 初回プロセス

DSPL4DevOps の初回プロセスの各工程の目的と実施内容について述べる。

不確実性分析工程： システムの開発ならびに運用において必要な情報のうち，開発者または運用者によって認知され，定義され，かつ開発時において未解決となっているものを不確実性と呼ぶことにする。この工程では，不確実性を含んだ要求から，システムの開発ならびに運用に係る不確実性を洗い出し，不確実性を解決する手段を検討し，それらを意思決定の木としてモデリングする。また，検討する各々の不確実性の解決手段について，その解決手段を採った際のシステムそのものの改変，システムのコンフィギュレーション変更，運用プロセスへの影響と対策を，不確実性影響評価／対策表にまとめる。この工程では，要求から不確実性を取り除き，システム運用分析工程に渡す。不確実性分析について詳細は著者による文献 [5] を参照されたい。

フィーチャ分析工程： 開発するシステムのフィーチャ分析を実施し，フィーチャモデルを作成する。フィーチャ分析の結果，見出されたシステムのフィーチャすべてを実装する訳ではないことに注意されたい。フィーチャ分析工程の目的は，開発時においてはシステムに必要とされる動的変更の範囲を定めると同時に，システムの諸機能の抽象度と関心事の分離を進めて改変に強いアーキテクチャを定義できるようにすること，また運用時においてはシステムのコンフィギュレーションの記述と管理を機械的に行えるようにすることにある。

システム運用分析工程： システムの要求を分析し，システム自身の振舞いによって実現する要求とシステムの運用によって実現する要求とに分け，前者をシステム開発要求として，後者はシステム運用プロセスに詳細化する。DevOps では，システムをモニタリングし，その運用データに基づいてシステムそのものや運用の改善を行う。システム開発要求には，システム運用のモニタリングに関する要求も含まれていることに注意されたい。

システム開発分析・設計・実装工程： システムそのものの実現に係る分析を行ってシステム要求仕様書を作成し，それに基づいてシステムの設計を行って外部設計書，内部設計書を作成し，動的適応システムの実装を行う。

運用： システム開発分析／設計／実装工程で実現された動的適応システムを，システム運用プロセスに沿って運用する。システム運用の際生じた要求のうち，システムのコンフィギュレーション変更で実現できる要求はそれで対

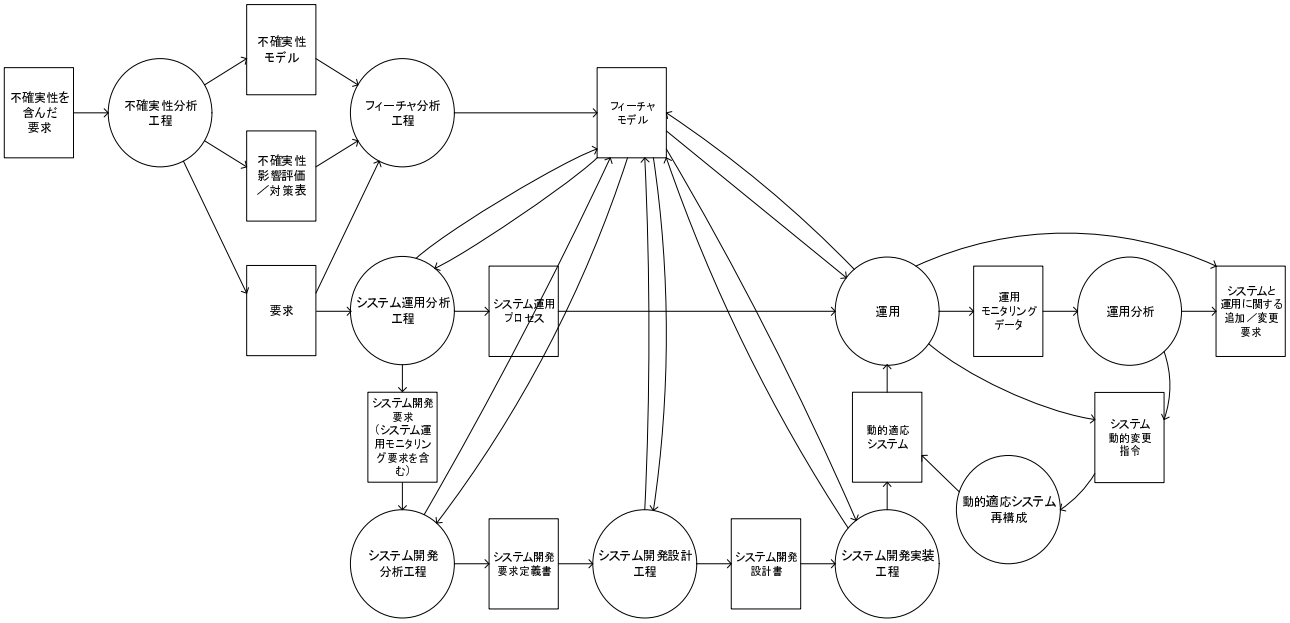

図 1 DSPL4DevOps 初回プロセス

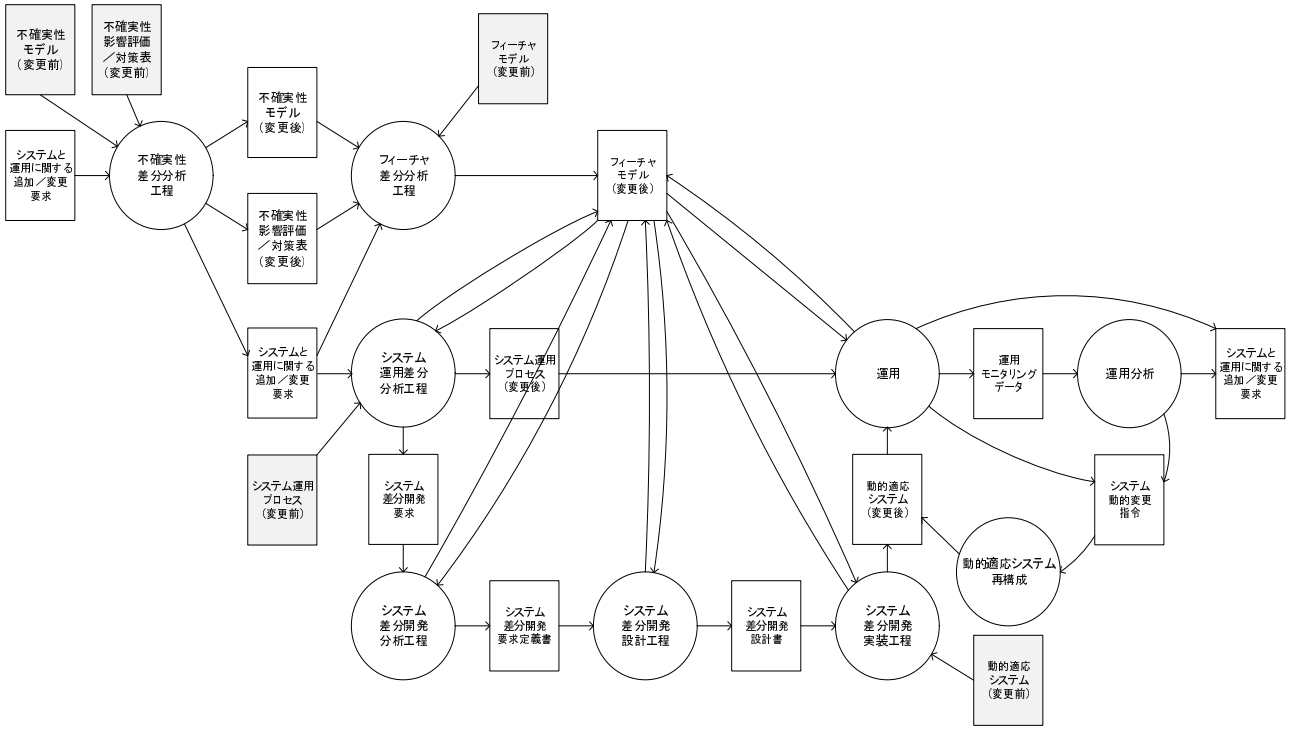

図 2 DSPL4DevOps 改善プロセス

応し，そうでないものはシステムと運用に関する追加／変更要求に挙げる。

運用分析： DevOps ではシステムの運用をモニタリングし，きめ細かなシステム運用の最適化が図られる。運用者は運用時に得られた運用モニタリングデータを分析し，システムの動的変更を行ってシステム運用の最適化を図るか，それができない場合はシステムと運用に関する追加／変更要求に挙げる。

動的適応システム再構成： システム運用の際生じた要求，あるいは運用分析の結果生じた要求のうち，不確実性分析工程において運用時の不確実性として想定され，かつシステムのコンフィギュレーション変更で実現できるようにしていたものについては，直ちにシステムのコンフィギュレーション変更を行って対処する。

3.3 改善プロセス

DSPL4DevOps の改善プロセスの各工程の目的と実施内容について述べる。改善プロセスでは，初回プロセスで構

築され，0 回以上の改善プロセスの適用で改善されてきた動的適応システムの運用で挙がった，システムと運用に関する追加／変更要求を実現すべく，動的適応システムそのものとその運用プロセスの改善を図る。

このプロセスでは，（当然のことながら）動的適応システムならびにその運用プロセスを最初から作り直すことは行わない。まず，システムと運用に関する追加／変更要求を，システムに対する追加／変更要求と運用プロセスに対する追加／変更要求とに分ける。前者は要求から実装に至る差分開発プロセスで実現する。これは動的適応システムを派生開発することに相当する。また，後者はシステム運用プロセスの改訂によって実現する。

改善プロセスでは，システムと運用に関する追加／変更要求に対して不確実性ならびにフィーチャの差分分析を実施し，不確実性モデルと不確実性影響評価／対策表，ならびにフィーチャモデルの改訂を行う。

3.4 フィーチャモデルと不確実性モデル

一連の工程において，フィーチャモデル，ならびに不確実性モデルが全てのドメインエンジニアリング（開発）とアプリケーションエンジニアリング（運用）を明に暗に支配するファーストクラスモデルとなる。

フィーチャモデルはシステムの動的変更の範囲を規定し，また各工程で生産される動的適応システムの実現に関する成果物，ならびに動的適応システムの運用プロセスに関する成果物について，抽象度をまたがって紐付ける索引となる。動的システムの運用においてシステムを動的変更する際の機能の有効化／無効化，パラメータ設定等のコンフィグレーションはフィーチャを単位に行われる。すでに述べた通り，動的適応システム開発時に全てのフィーチャが実現されるのではなく，動的適応システムの運用開始後，必要になってから実現されるフィーチャもあり得る。DevOps の観点からは，開発担当と運用担当との対話で見出された将来必要とされ得る機能，あるいはあり得る動的変更の要素を，開発初期のうちからフィーチャモデルとして記述しておくことで，将来の変化に迅速に対応できるようにする効果を期待している。

フィーチャモデルには，動的適応システムの何をコンフィギュレーションできるかは表現されているが，どのような場合にどのようにコンフィギュレーション変更をすべきかは表現されていない。不確実性モデルには，開発段階で開発者と運用者から認知された動的適応システムの開発時ならびに運用時に係る不確実性とそれらに対する考えられる解候補が記述され，不確実性影響評価／対策表には各解候補を選択したときの影響と対策が記述される。動的適応システムのコンフィギュレーション変更で対応できる不確実性については，不確実性影響評価表の対策欄に，フィーチャ選択の方法としてコンフィギュレーション変更の内容を記述する。すなわち，運用においてどのような場合にどうようにコンフィギュレーションすべきかは，不確実性分析工程において不確実性影響評価／対策表に記述される。システム運用分析工程では，この不確実性影響評価／対策表を参照して，システム運用プロセスを作成する。

表 1 に不確実性影響評価／対策表の例を示す。この表には GUI アプリケーションのレイアウト制御において未知の画面サイズの端末が使用された場合の解決策とその影響，開発時と運用時の対策を書いている。

3.5 動的適応システムの構成要素

動的 SPL のドメインエンジニアリング工程で開発される動的適応システムには下記の機構を設ける。

- **フィーチャモデルデータベース**： フィーチャモデルを格納するデータベースである。
- **コンポーネントリポジトリ**： システムを構成するコンポーネント（コード資産）を格納するリポジトリである。リポジトリに格納されるコンポーネントには識別子が与えられる。
- **リソースリポジトリ**： システムで参照あるいは更新されるリソース（データ資産）を格納するリポジトリである。リポジトリに格納されるリソースには識別子が与えられる。
- **ソフトウェア動的再構成機構**： ソフトウェア動的再構成機構は，運用者あるいは運用モニタリング機構からの再構成指令を契機に運用中のシステムを再構成する。再構成指令には新構成のシステムのフィーチャ選択が含まれる。動的再構成機構はフィーチャモデルデータベース中のフィーチャモデルを参照し，再構成指令によって与えられたフィーチャ選択を検証し，フィーチャに紐付けられた再構成プロセス記述に従って，コンポーネントリポジトリ中のコンポーネントを組み合わせてシステムを再構成する。
- **リポジトリ更新機構**： 開発者ならびに運用者からの更新指令に基づき，コンポーネントリポジトリならびにリソースリポジトリ中のコンテンツを生成，更新，改名，削除する。
  - **生成操作**：コンポーネント／リソース中に新たに格納するコンポーネント／リソースのために識別子を発行する。生成操作で行うのは，識別子の発行のみであり，内容そのものの追加は更新操作で行う。
  - **更新操作**：コンポーネント／リソースリポジトリ中のすでに識別子の与えられているコンポーネント／リソースの内容の更新を行う。更新はネットワークから直接内容の全体，あるいは差分をサーバより配送することで行う。さらに，リソースの更新については，コンポーネントリポジトリ中の識別子で指定されるコンポーネントの実行によっても可能とする。

表 1 不確実性影響評価／対策表の例

不確実性	解候補	影響	開発時の対策	運用時の対策
想定しない画面サイズ	対角線長が最も近いサイズ向けのレイアウトを適用し画面を縮小する。	文字がつぶれて読めないかもしれない。	画面長押し&ドラッグで画面を上下左右にスクロールできるようにする。	フィーチャ「全体縮小表示」を選択して再コンフィギュレーション。未知サイズとして開発者にメッセージを送り新レイアウトの追加開発を促す。
	対角線長が最も近いサイズ向けのレイアウトを適用し画面をスクロールできるようにする。	見た目が悪い。スクロールが面倒かもしれない。	画面長押し&ドラッグで画面を上下左右にスクロールできるようにする。	フィーチャ「全体スクロール表示」を選択して再コンフィギュレーション。未知サイズとして開発者にメッセージを送り新レイアウトの追加開発を促す。

- **改名操作**：コンポーネント／リソースリポジトリ中のすでに識別子の与えられているコンポーネント／リソースの識別子を付け替える。
- **削除操作**：コンポーネント／リソースリポジトリ中のすでに識別子の与えられているコンポーネント／リソースを削除する。

いずれの操作に関してもリポジトリ更新機構は明示的に設けられる同期点へのロールバックを保証する。

- **運用モニタリング機構**：コンポーネントから送られるイベントやシステムの基盤ソフトウェアから得られるデータを記録，加工し，サーバへ転送する。また，動的適応の必要を判断し，ソフトウェア動的再構成機構に再構成指令を送る。運用モニタリング機構については，それ自身，コンポーネントリポジトリ中のコンポーネントで構成し，動的再構成を可能にする。

4. まとめ

本稿では，動的 SPL を DevOps の一実現手段として捉え，不確実性分析の結果に基づいて，動的適応システムとその運用プロセスを構築するプロセス，ならびに運用時に挙がった動的適応システムとその運用に対する追加／変更要求を成長的な派生開発によって実現するプロセスを提唱した。これらのプロセスを DSPL4DevOps と総称し，前者をその初回プロセス，後者をその改善プロセスと呼ぶ。DSPL4DevOps では，不確実性モデルおよびフィーチャモデルがファーストクラスモデルとして扱われる。不確実性モデルは動的適応システムの開発と運用に係る不確実性への対処の検討，ひいては動的適応システムの動的適応性と運用に係る要求の基盤となる。フィーチャモデルは動的適応システムのコンフィギュレーションの記述と管理の基盤となる。また，このプロセスを実現するうえで動的適応システムに必要となる諸機構についても述べた。

本稿は，ケーススタディを伴う DSPL4DevOps の妥当性の評価には至らず，動的 SPL と DevOps の諸概念の整理，ならびに動的 SPL および DevOps への著者が提唱する不確実性分析手法の導入に留まっている。今後，車載アプリケーションを題材に DSPL4DevOps の妥当性の評価を進めたい。

謝辞 本研究は JSPS 科研費 15H05708 の助成を受けたものです。

参考文献

[1] Klaus Pohl, Günter Böckle, and Frank van der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*, Springer 2005.
[2] Svein Hallsteinsen, Mike Hinchey, Sooyong Park, and Klaus Schmid, “Dynamic Software Product Lines,” *Computer*, Vol. 41, No. 4, pp. 93–95, Apr. 2008.
[3] Mike Hinchey, Sooyong Park, and Klaus Schmid, “Building Dynamic Software Product Line,” *Computer*, Vol. 45, No. 10, pp. 22–26, Oct. 2012.
[4] Len Bass, Ingo Weber, and Liming Zhu, *DevOps: A Software Architect’s Perspective*, Addison-Wesley, 2015.
[5] 中西 恒夫, 馬 立東, 久住 憲嗣, 福田 晃, 「システム開発のための不確実性フレームワーク」, 情処研報, Vol. 2014-EMB-32, No. 6, 2014 年 3 月.
[6] Luciano Baresi, Sam Guinea, and Liliana Pasquale, “Service-Oriented Dynamic Software Product Lines,” *Computer*, Vol. 45, No. 10, pp. 42–48, Oct. 2012.
[7] Jaejoon Lee, Gerald Kotonya, and Daniel Robinson, “Engineering Service-Based Dynamic Software Product Lines,” *Computer*, Vol. 45, No. 10, pp. 49–55, Oct. 2012.