HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

«科学的モデリングシリーズ»

科学的SW-Plug＆Play
と
科学的SW-バリアント・デベロップメント
(ソフトウエア派生開発)
Part1

HASHIMOTO SOFTWARE CONSLUTING INTERNATIONAL INC.

SEI Partner
CMMI

グローバルな大競争時代

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

企業の課題

- 市場投入の短い開発や生産期間
- 多品種のシステムの開発や生産(多品種少量開発)
- 価格競争
- 大規模化複雑化する製品やシステムの品質確保
- 増加するオフショア開発／アウトソ-シング開発への対応

科学的開発アプローチとツールによる自動化

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

課題を解決する科学的なアプローチ

- 『ソフトウエア科学』に立脚した極めて効率的な開発手法とプロセスの採用
- 企業の『ソフトウエア資産』の活用を最大化する
- 『最大のコスト』である人件費を大きく削減を実現するツールによる作業の自動化

ツールによる作業の自動化

- 仕様書・設計書などの文書の自動生成
- ソースコードの自動生成
- テストの自動化
- モデルやコードの検証(verification)および妥当性確認(validation)
- データー管理や構成管理などの管理活動の自動化
- 品質や生産性のメトリクス収集と分析の自動化

科学的な開発プロセスとツールによる自動化の時代

『科学的SW-Plug＆Play』と『科学的SW-バリアント・デベロップメント』

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

『ソフトウェアPlug and Play』
&
『ソフトウエア・バリアント・デベロップメント(ソフトウェア派生開発)』

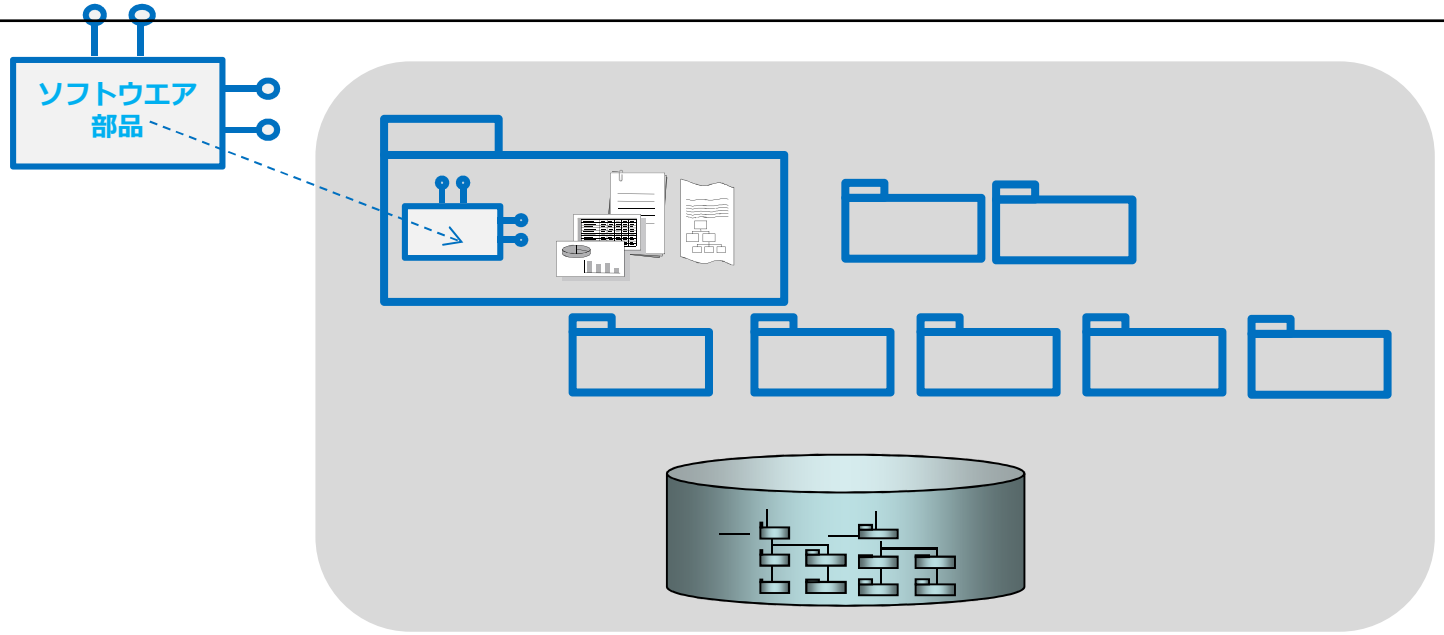
- 『ソフトウェア部品』を実現しソフトウエアの『プラグイン開発』が可能
- 生産性と品質が大きく向上する
- 現代の企業の開発スタイルに最適な開発アプローチ
  - 「ソフトウエア資産の再利用」
  - 「多品種少量生産型ソフトウェア開発」
  - 「オフショア開発」／「アウトソ-シング開発」
- 『科学的ソフトウエアCAD(モデリングツール)』による正確なモデル作成と自動化

『先進的構成管理技法』と 完全自動コンパイル・リンク&デプロイメント

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

『先進的構成管理技法』による完全自動コンパイル・リンク&デプロイメント

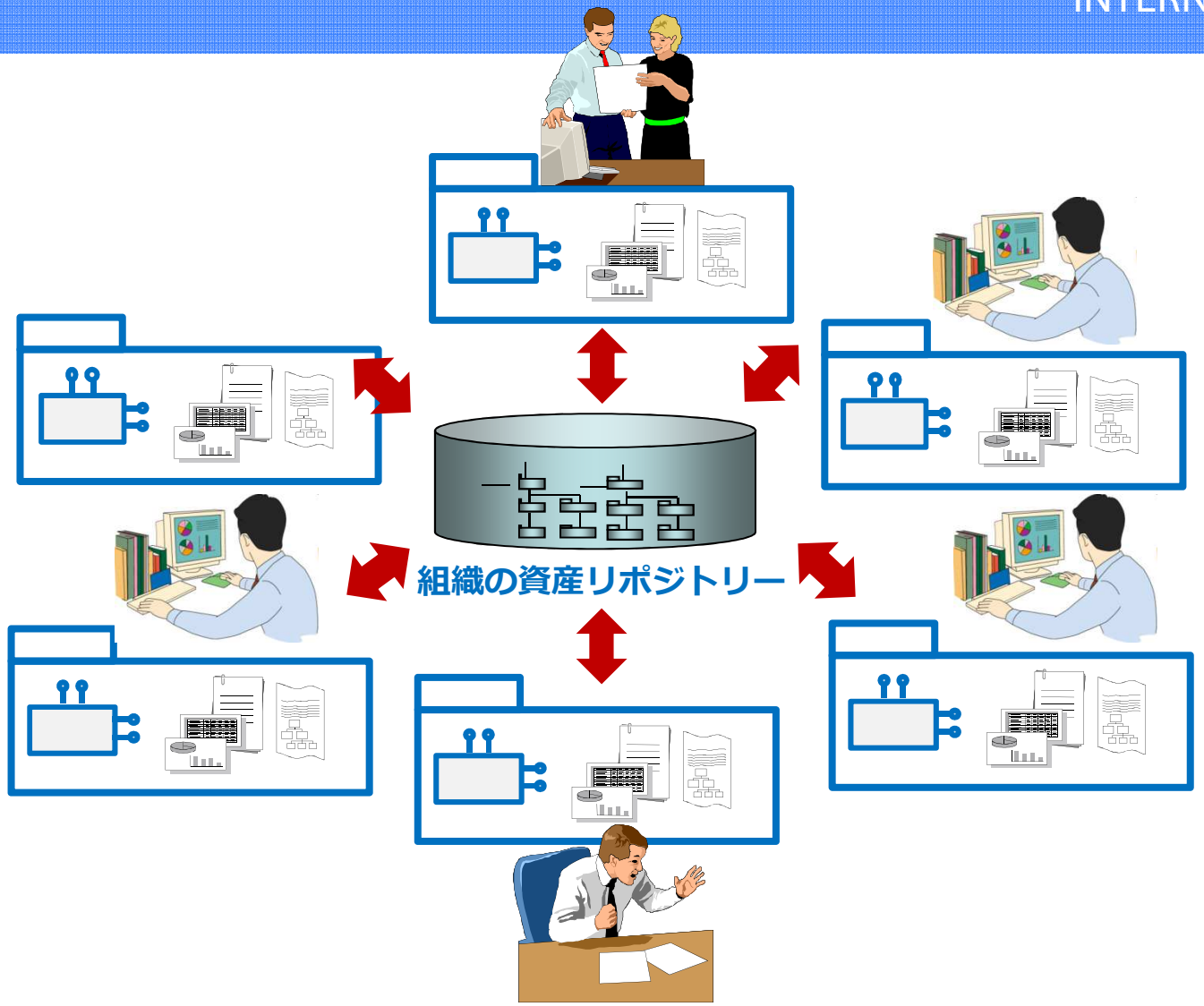
- 欧米の制御系のソフトウエア開発企業は,1000以上のバリアントの選択と組合わせ可能でデプロイメントまでが**完全自動化**されている
  - ➔瞬時にバリアントに対応したコードの選択とコンパイル・リンクおよびデプロイメンントが完了する
- **オブジェクト指向技術**と**モデリング技法**および**構成管理**の連携が重要



組織の資産リポジトリー

分散並行開発と完全自動デプロイメントを実現する『先進的構成管理』

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.



安全な『科学的SW-Plug＆Play』と『科学的SW-バリアント・デベロップメント』

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

『ソフトウェアPlug and Play』&『ソフトウエア・バリアント・デベロップメント』はソフトウエアの信頼性を向上させる

- 『ソフトウエア科学』を応用して設計モデルとコードの**信頼性**を高めている
  - 「タイプ(型)置換原理」による型安全な多相(ポリモフィズム)の実現
  - 「契約による設計」による型安全なクラスの設計と実装
  - 「演繹的型推論」による型安全なクラスの設計と実装
  - 「演繹的モジュラー推論機能」による安全な「派生開発」と「再利用」の実現
  - 「強い型付け言語」の利用による型安全なコードの作成
  - 「防衛的設計と実装」による堅牢なシステムの実現
- 『ソフトウエア科学』を利用することは「新規開発」だけでなく「バリアントデベロップメント(派生開発)」「ソフトウエア資産の再利用」にも重要となる

「品質」「生産性」「安全性」が確実に達成できる

目の前にある危機～
危険で間違った「派生開発」と「再利用」

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

“非”科学的な再利用や派生開発のアプローチは悲惨な結果招く

- ソフトウエアの再利用が原因の大参事としてAriane5がある
- 事故原因調査結果[Lio96]：
  - スローされた例外をキャッチできない不具合がAriane4のソフトウェアコンポーネントあった
  - このコンポーネントの再利用が原因で宇宙船のソフトウエアが破壊された
- JzquelとMayerは,事故の根本原因はコンポーネントの「契約」が存在しない事と主張[JM97]
  - **「契約」**とはソフトウエア科学を利用した操作・クラス・コンポーネントの「明確な意味の定義」のことであり下記の「特性」で定義する:
    - **不変条件**
    - **操作の事前条件／事後条件／例外**
- JzquelとMayerは以下のように結論付けている:
  - クラスやコンポーネントの「契約」を定義せず「タイプ(型)置換原理」を満足しない「派生開発」と「再利用」は**完全な愚行**である

日本は現在も**『間違った危険な開発』**が主流であり横行している

安全な派生開発・再利用を実現する重要原理 ～『開放閉鎖原理』

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

科学的モデリング規則:『開放閉鎖原理(OCP:Open-Closed Principle)』

- クラス(orコンポーネントorモジュール)は**「拡張可能で開放されている」**
- クラス(orコンポーネントorモジュール)は**「閉鎖されている」**のでクライアントから利用可能であること

『開放閉鎖原理(OCP:Open-Closed Principle)』By Robert C.Martin

- 『開放閉鎖原理』を適用すれば,すでにモジュールがコンパイルされてバイナリ形式になっているものは,それがリンクされたライブラリーであろうと,DLLであろうと,Javaの.jarファイルであろうと手を触れる必要がない
- 『開放閉鎖原理』はオブジェクト指向設計の核心である
- 『開放閉鎖原理』に従う事でオブジェクト指向技術から得られる最大の利益を享受できる
  - 柔軟性
  - 再利用性
  - 保守性

継承による『開放閉鎖原理』の実現

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

『開放閉鎖原理(OCP:Open-Closed Principle)』By Bertrand Meyer
• 『開放閉鎖原理』を可能とするのは継承機能だけである

『開放閉鎖原理(OCP:Open-Closed Principle)』
• 『正しい設計には,継承を正確に使用する事が極めて重要である』by Ian Joyner • 『OO開発の威力の多くは継承の重要な概念によるものである』by Herbert Toth

重要事項～「クラス」と「タイプ(型)」の区別

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

クラスとタイプ(型)の区別とタイプ(型)置換原理

- **サブクラス(subclass)**と**サブタイプ(subtype)**は明確に異なる概念である
- 継承階層内でサブクラスは任意のスーパークラス**『タイプ(型)置換原理』**を満たす必要がある
  ➔**「振る舞いサブタイプ(型)」**
- **「振る舞いサブタイプ(型)」**
  ➔サブクラスのオブジェクトをスーパークラスのオブジェクトの**意味(振る舞い)を保証して代わりに利用できる「部分型」**

有名な『タイプ置換原理』の１つ～『リスコフ置換原理』

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

サブクラスが**『振る舞いサブタイプ(型)』**となる規則
クラスやコンポーネントの「派生開発」と「再利用」を**安全**に行える

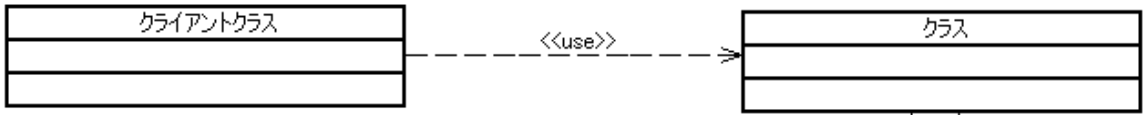
リスコフ置換原理[Liskov et al.,1994]の意味

- スーパータイプの**不変条件(invariant)**はサブタイプにおいて＜強める＞ことはできるが＜弱める＞ことはできない
  - **属性不変条件**
  - **クラス不変条件**
  - **$SC_{invariant} \Rightarrow Invariant$**
- スーパータイプの**操作の事前条件(pre-condition)**はサブタイプにおいて＜弱める＞ことはできるが＜強める＞ことはできない
  - **$C_{pre} \Rightarrow SC_{pre}$**
- スーパータイプの**操作の事後条件(post-condition)**はサブタイプにおいて＜強める＞ことはできるが＜弱める＞ことはできない
  - **$SC_{post} \Rightarrow C_{post}$**
  - **$(C_{pre} \wedge SC_{post}) \Rightarrow C_{post}$**

『開放閉鎖原理』『振る舞いサブタイプ(型)』『リスコフ置換原理』の効果①

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

- クラス「クライアント」は全く影響をうけず,「機能の拡張」「部品化」「派生開発＆再利用」を達成するために**『開放閉鎖原理』**を適用する

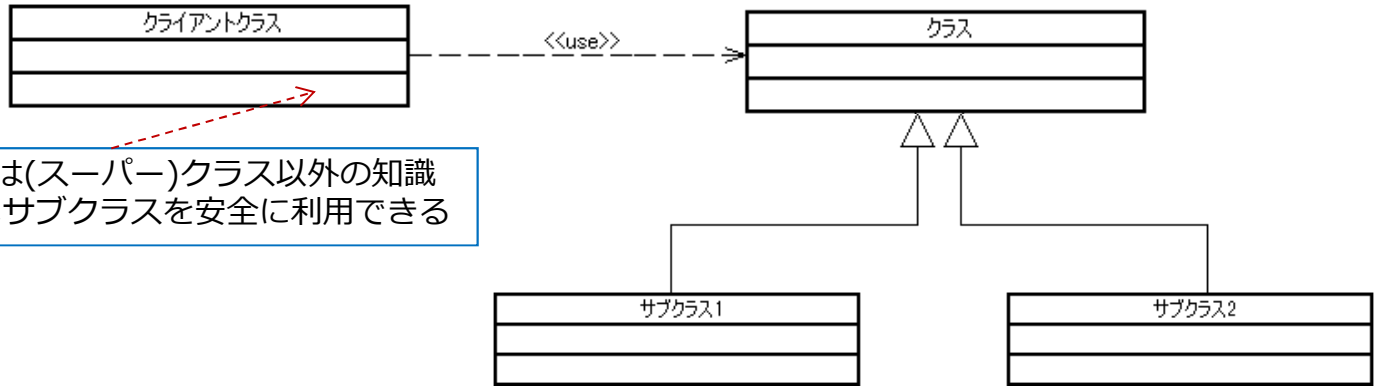


『開放閉鎖原理』『振る舞いサブタイプ(型)』『リスコフ置換原理』の効果②

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

- サブクラスで機能を拡張してもクラス「クライアント」は**全く影響をうけない**

- スーパークラスは**「仕様」**を定義するため抽象クラス



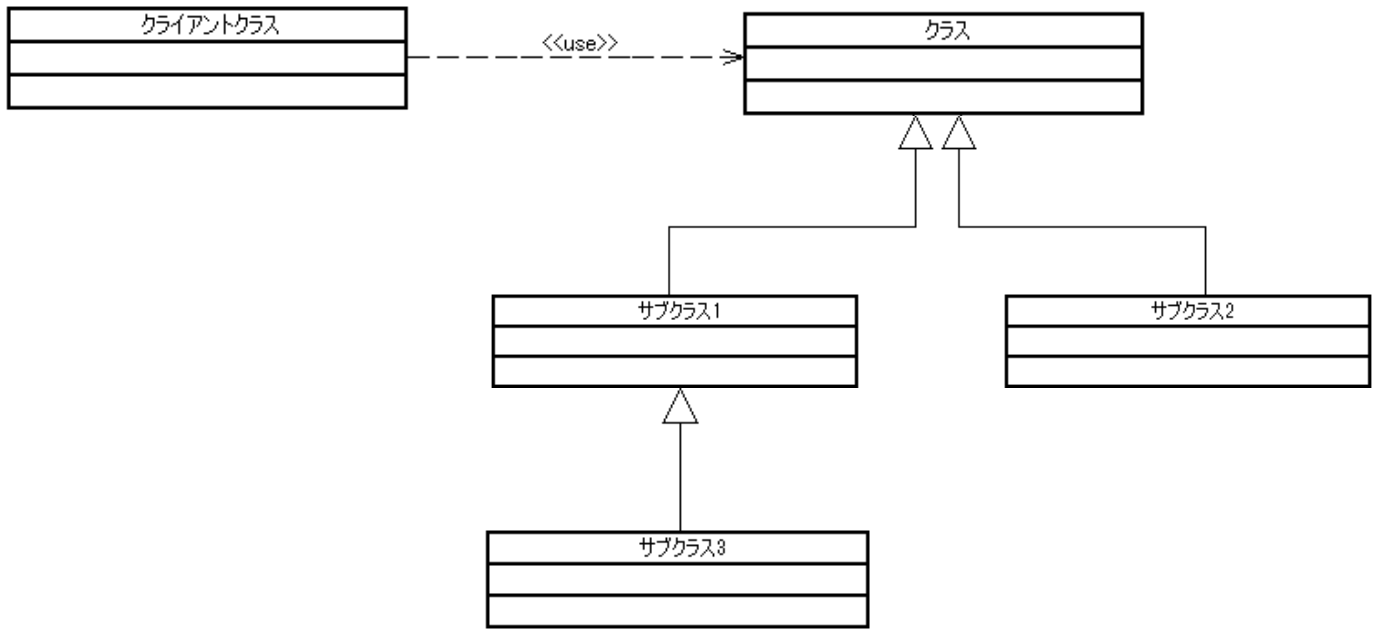
- クライアントは(スーパー)クラス以外の知識を持たないが,サブクラスを安全に利用できる

- サブクラスは**「仕様」に基づいて機能を実装する**

- **『開放閉鎖原理』**を適用するときは**『リスコフ置換原理』**を満足させ**『振る舞いサブタイプ』**が成立させれば,**サブクラスで継承した操作を再定義して機能を改定することが自由に可能**

『開放閉鎖原理』『振る舞いサブタイプ(型)』『リスコフ置換原理』の効果③

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

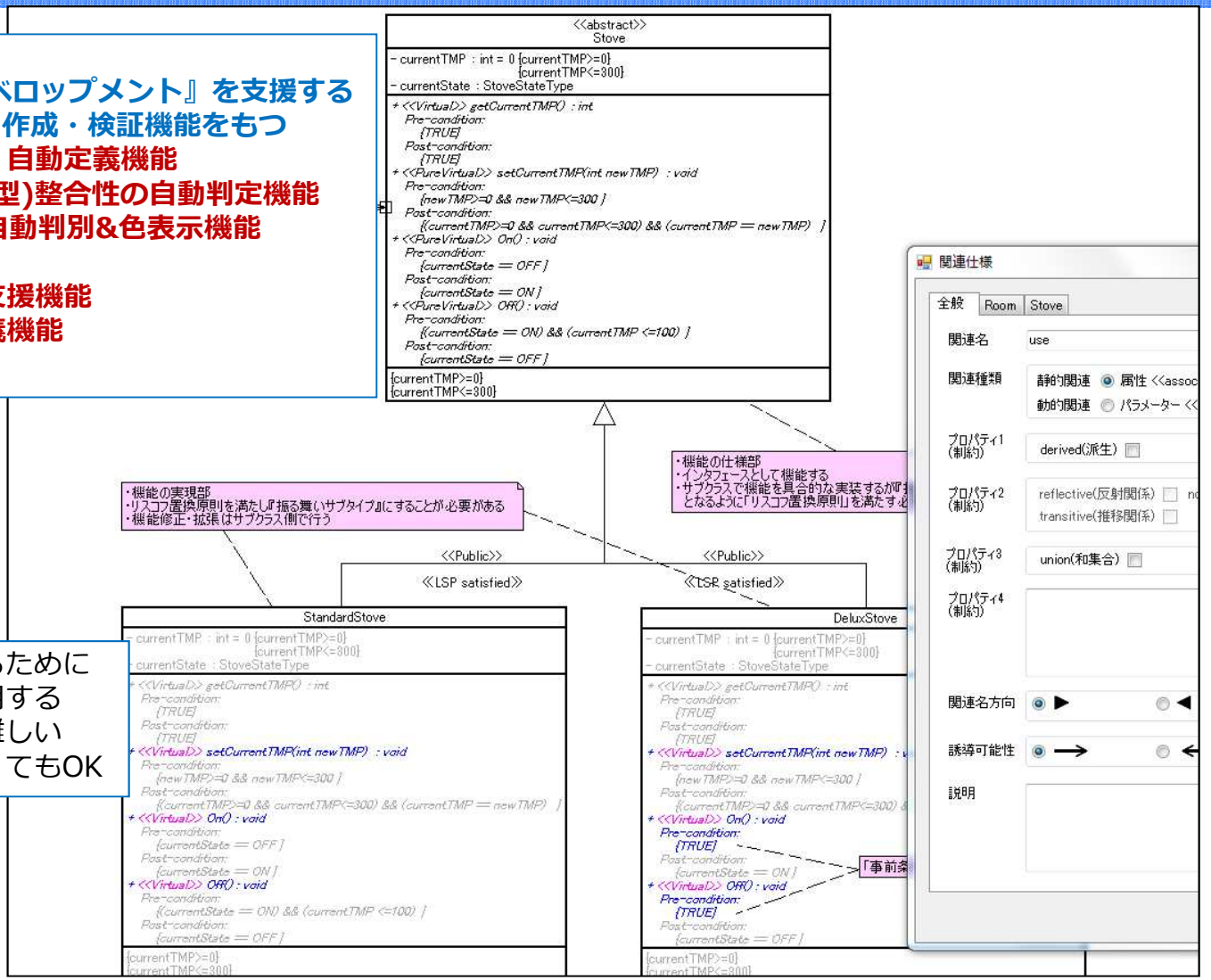


- **『開放閉鎖原理』**を適用するときは**『リスコフ置換原理』**を満足させ**『振る舞いサブタイプ』**
  を満たせば継承をどんどん深く出来き,機能拡張が思いのまま可能となる

『ソフトウエアCAD(科学的モデリングツール)』による作業の自動化

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

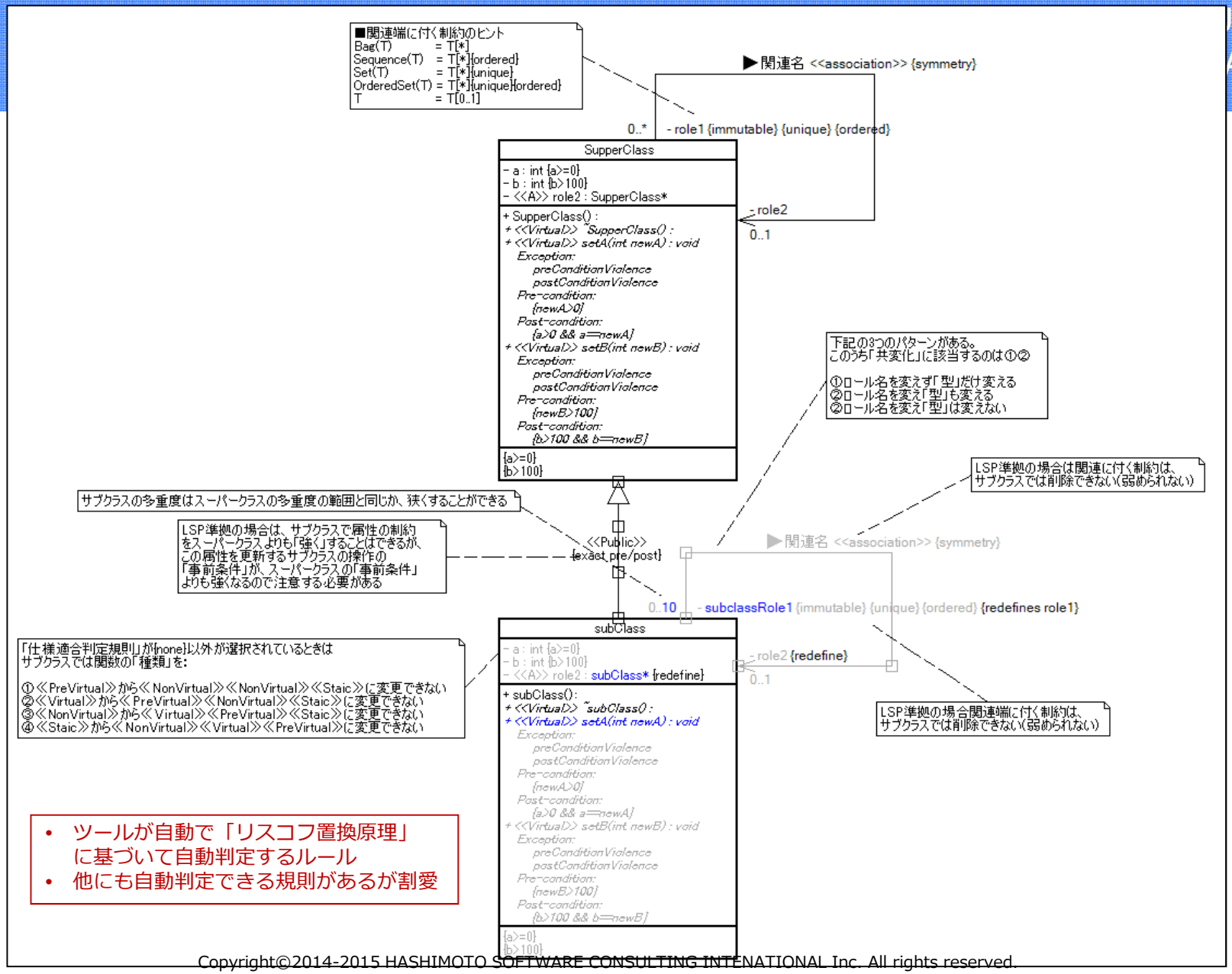
- 『科学的SW-Plug&Play』
  『科学的SW-バリアント・デベロップメント』を支援する
- 正確なモデルを迅速&自動的に作成・検証機能をもつ
  - 「振る舞いサブタイプ」自動定義機能
  - 演繹推論によるタイプ(型)整合性の自動判定機能
  - {redefine}{refine}自動判別&色表示機能
  - 自動コード生成
  - 自動ドキュメント生成支援機能
  - 静的関連・動的関連定義機能
  - etc



- 作業効率を劇的に向上させるために
  ツールによる自動処理を利用する
- ツールの機能でユーザーは難しい
  理論・原理に精通していなくてもOK

(＊)HSCI Modelling tool
特許出願済み／Patent Pending

ツールで『振る舞いサブタイプ(型)』の自動判定をおこなう①



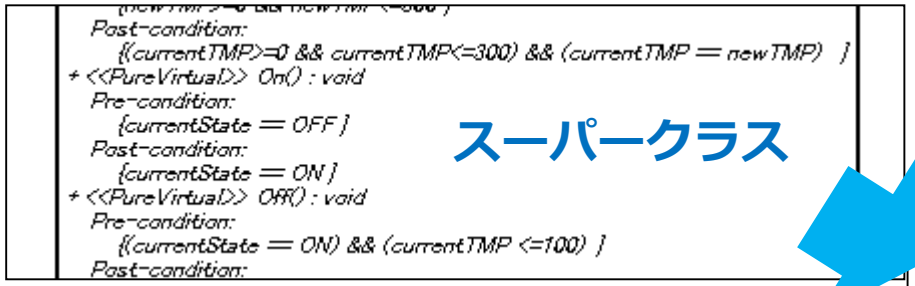
- ツールが自動で「リスコフ置換原理」に基づいて自動判定するルール
- 他にも自動判定できる規則があるが割愛

ツールで『振る舞いサブタイプ(型)』の自動判定をおこなう②

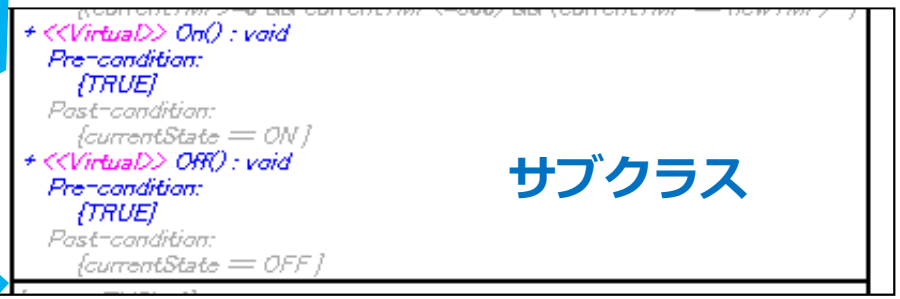
HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

ソフトウエアCAD(科学的モデリングツール)の演繹推論エンジン

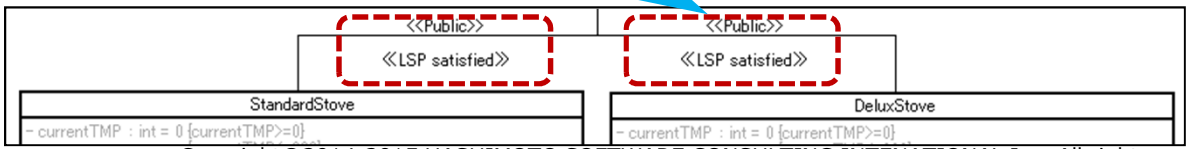
- 「クラスの表明(契約)」の表示と整合性判定および自動コード生成(C言語／C++／その他)
  - クラス不変条件
  - 操作の事前条件／事後条件
  - 操作の例外
- 「タイプ(型)置換原理」と「モジュラー推論」の数種類の演繹推論規則を実装し判定とコード生成が可能



- サブクラスのタイプ(型)の『**拡張**』『**特殊化**』を寸時にチェックし判定する
- 色表示で違いを表示

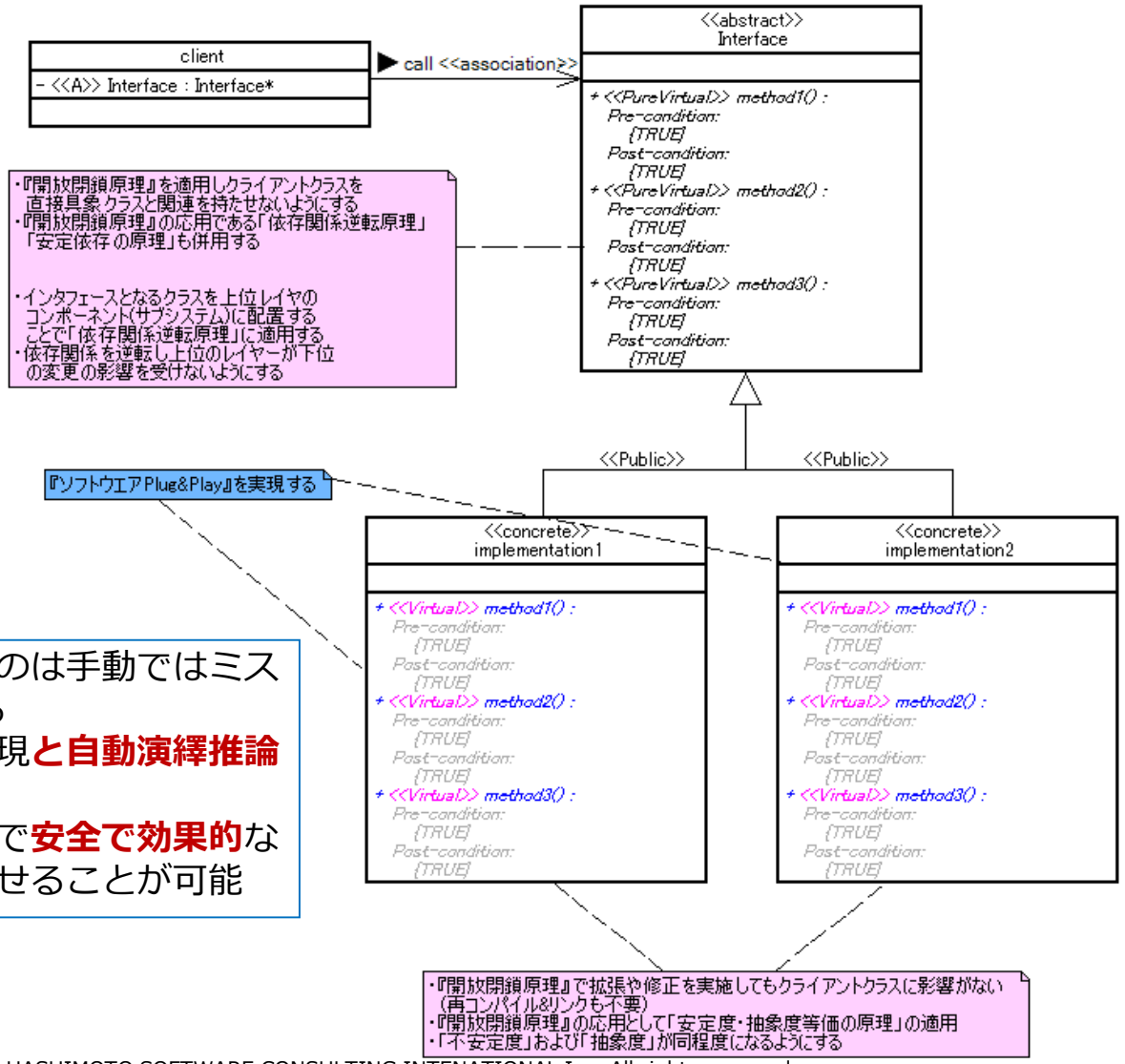


- ツール判定で『**リスコフ置換原理**』を満足すると『**振る舞いサブタイプ**』が成立し**«LSP satisfied»**と検討結果が表示される



ツールで『振る舞いサブタイプ(型)』の自動判定をおこなう③

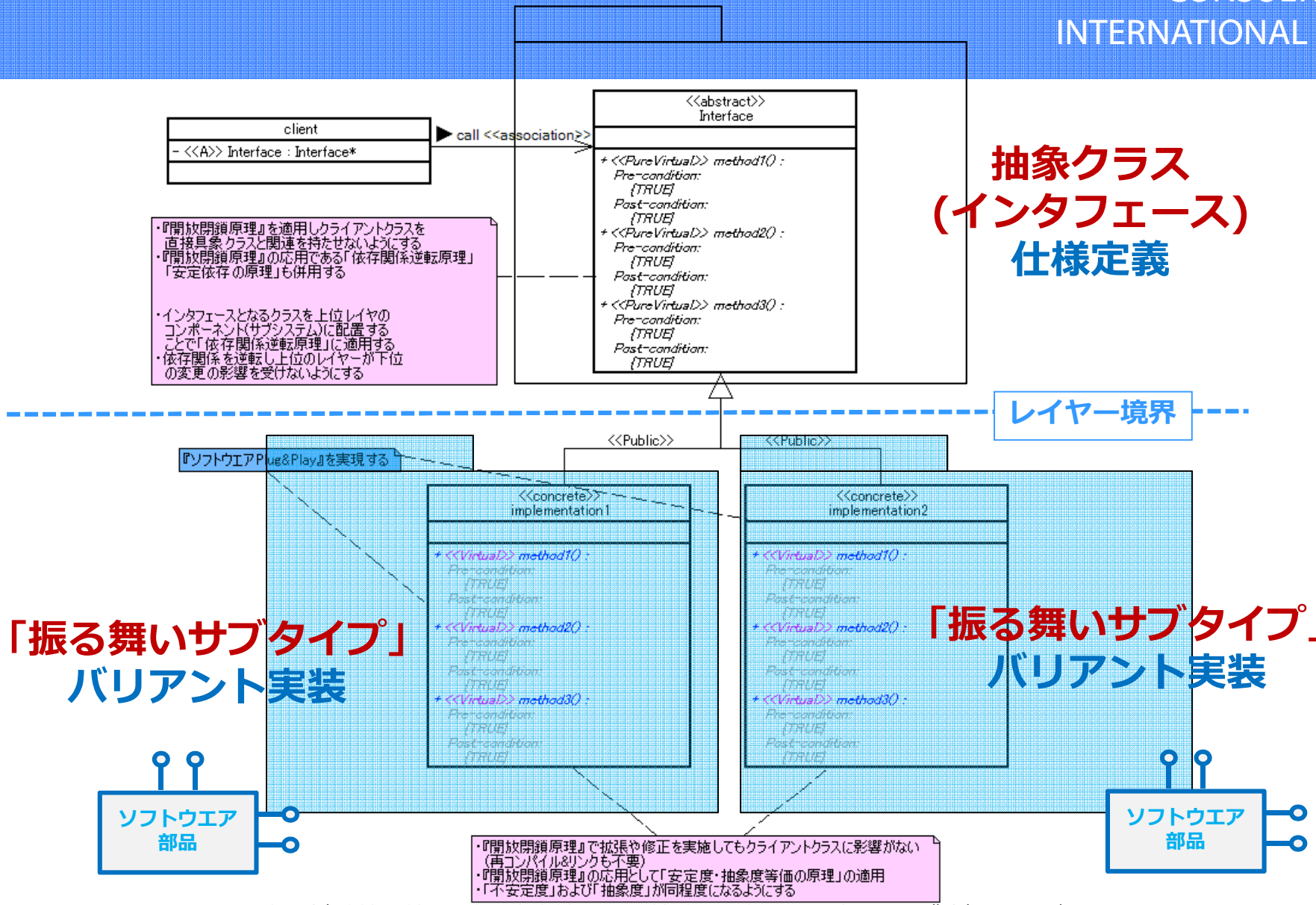
HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.



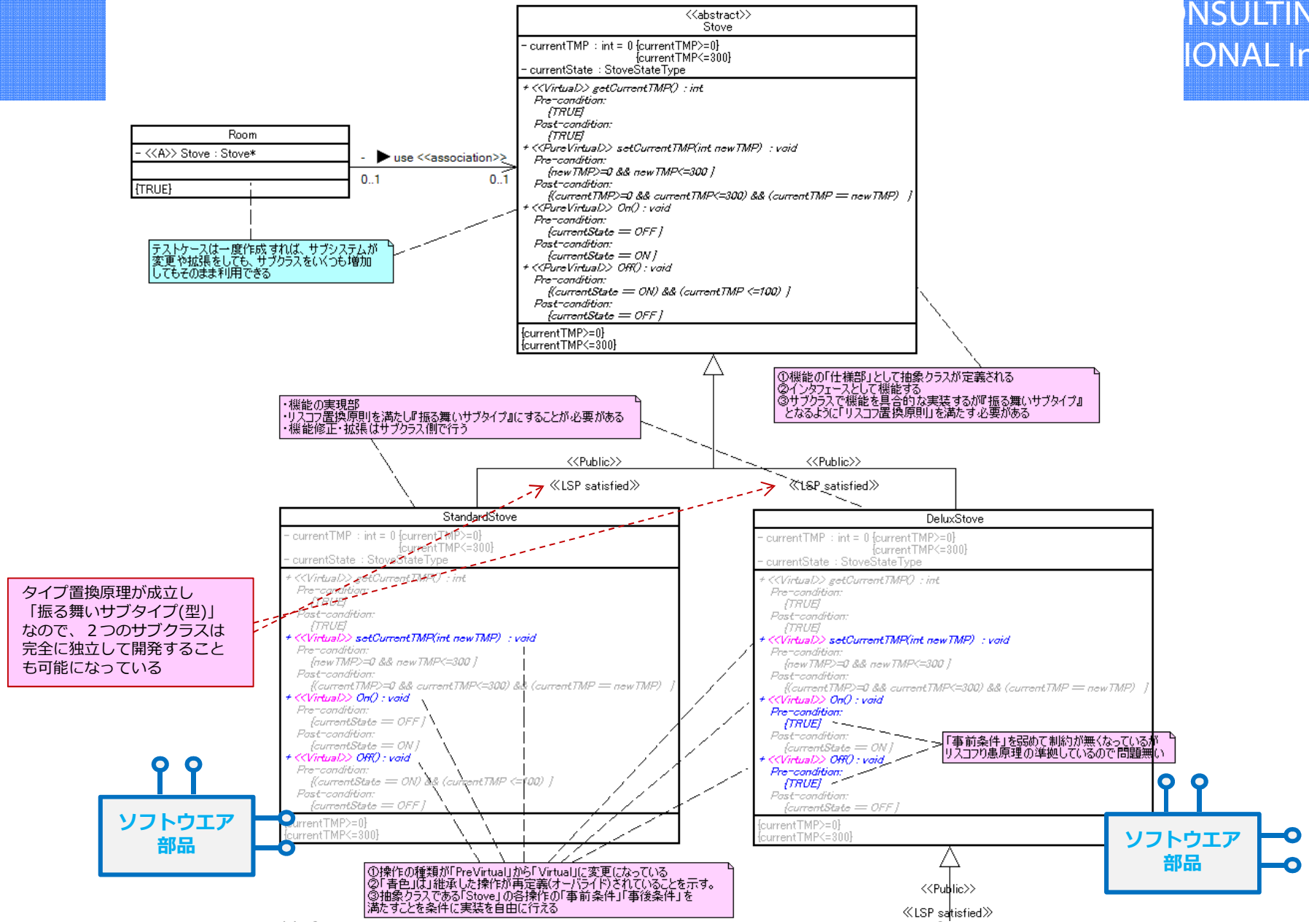
- 数多くのクラスを作成＆判定するのは手動ではミスが起きやすく,また非効率的である
- ツールによる**タイプ置換原理**の実現**と自動演繹推論**とで**安全な開発**を行う
- **振る舞いサブタイプ**を満たすことで**安全で効果的**な**派生開発**と**再利用**が格段に向上させることが可能

『仕様と実装の分離』＆『ソフトウエア部品』の基本原理①

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.



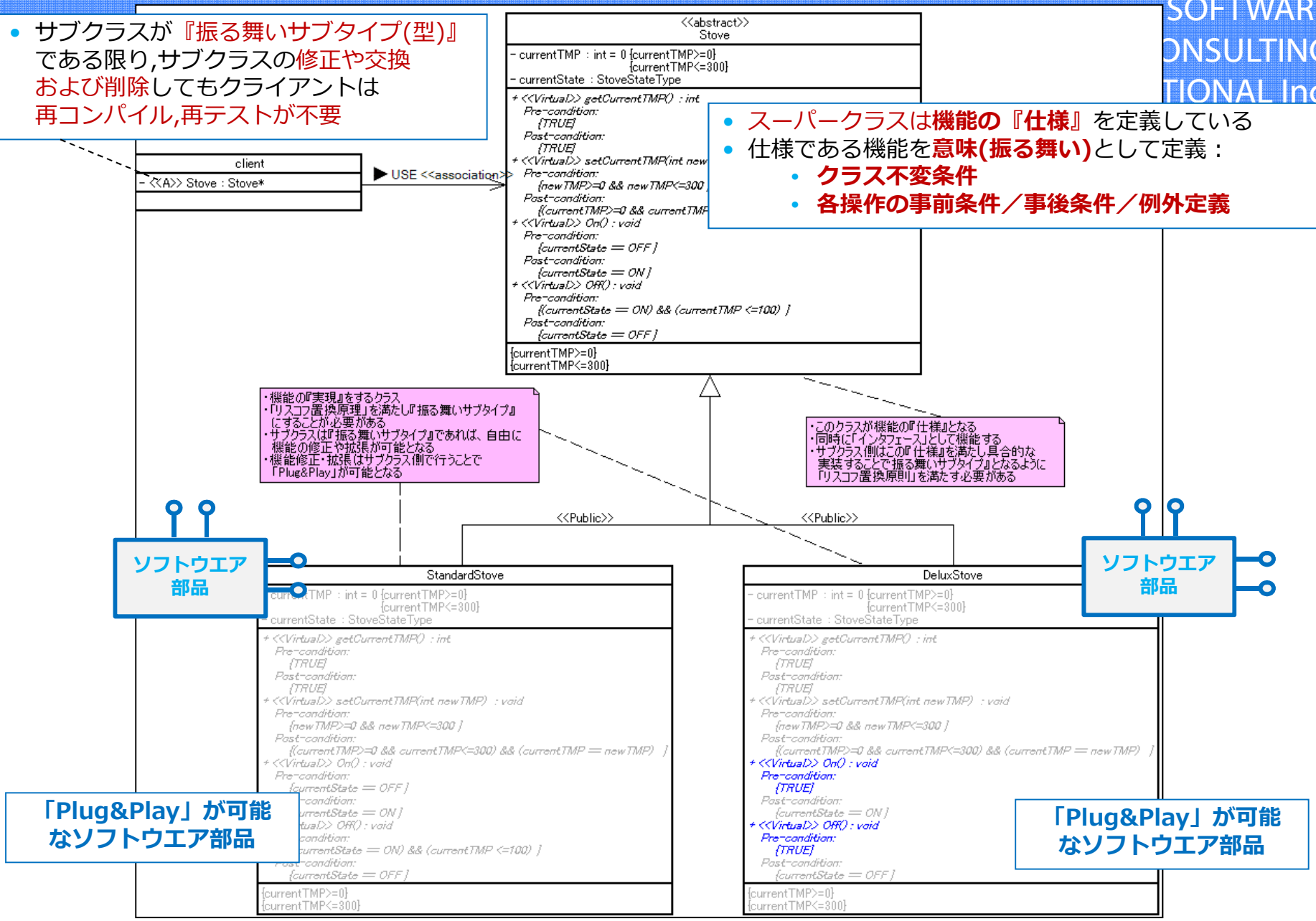
『仕様と実装の分離』＆『ソフトウエア部品』の基本原理②



『振る舞いサブタイプ』で「Plug&Play」を実現

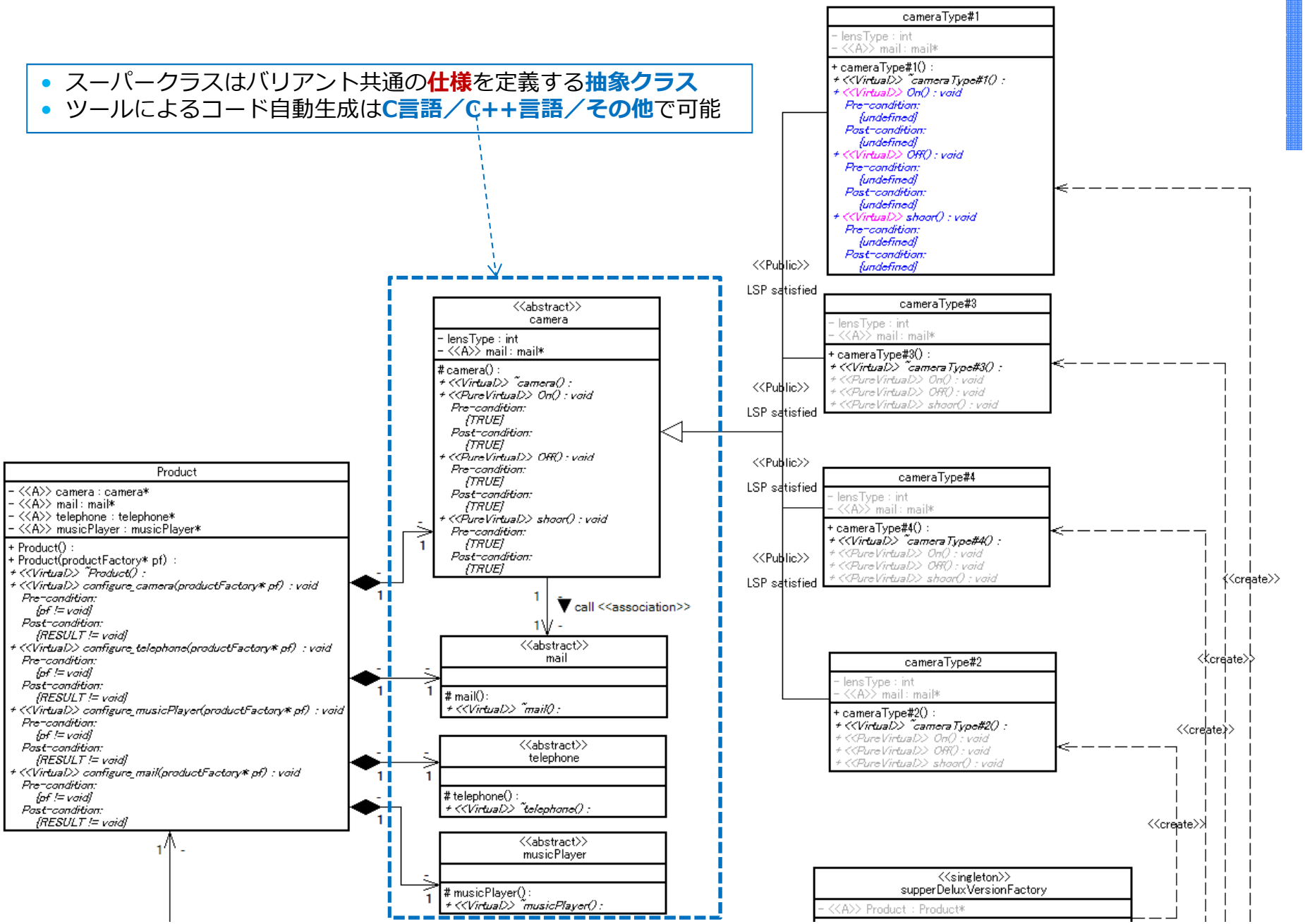
- サブクラスが『振る舞いサブタイプ(型)』である限り,サブクラスの修正や交換および削除してもクライアントは再コンパイル,再テストが不要

- スーパークラスは**機能の『仕様』**を定義している
- 仕様である機能を**意味(振る舞い)**として定義：
  - **クラス不変条件**
  - **各操作の事前条件／事後条件／例外定義**



バリアントの『仕様』を抽象クラスで定義する HASHIMOTO

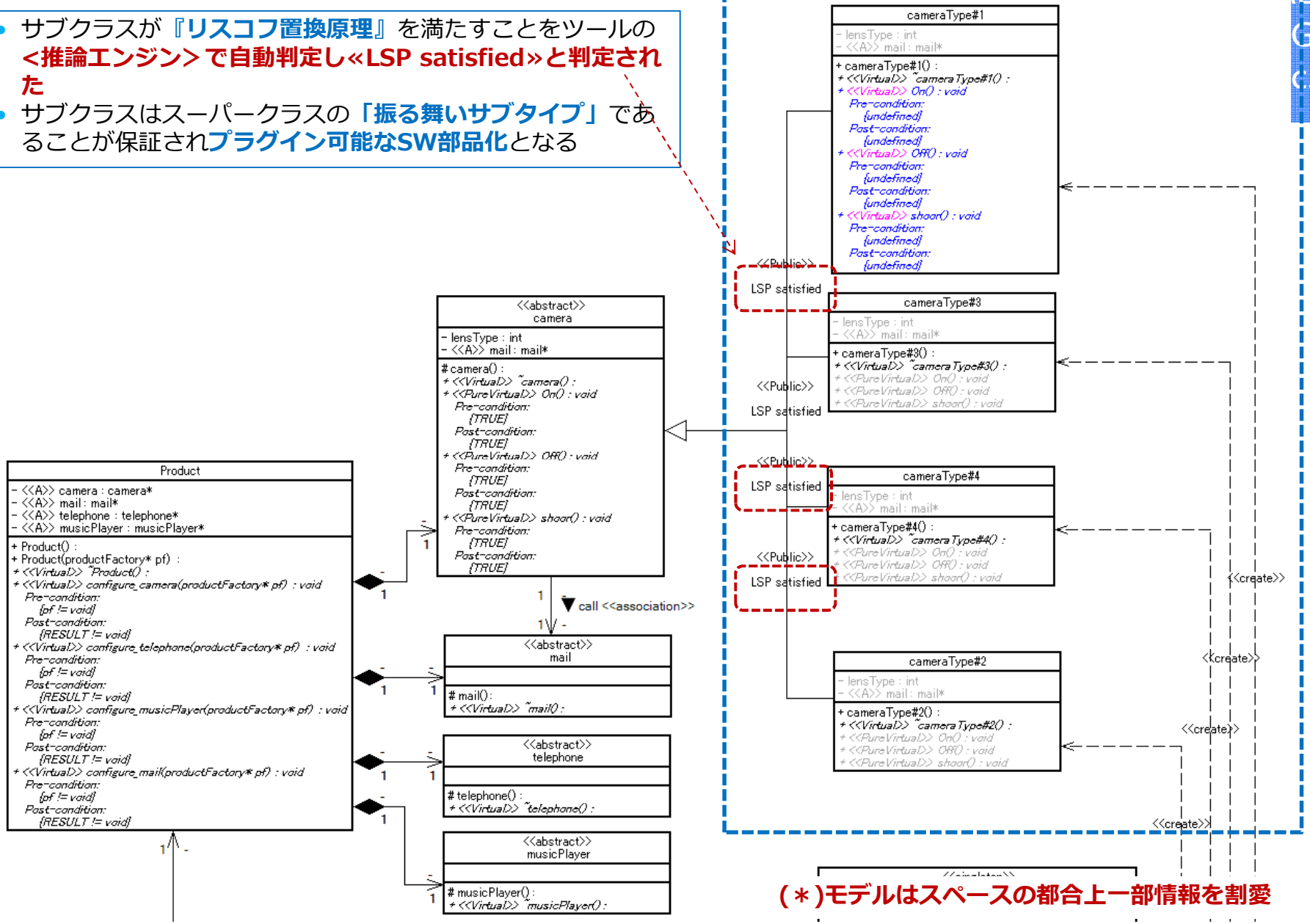
- スーパークラスはバリアント共通の**仕様**を定義する**抽象クラス**
- ツールによるコード自動生成は**C言語／C++言語／その他**で可能



バリアントを『振る舞いサブタイプ』で実装

HASHIMOTO SOFTWARE ENGINEERING

- サブクラスが**『リスコフ置換原理』**を満たすことをツールの**<推論エンジン>で自動判定し«LSP satisfied»と判定された**
- サブクラスはスーパークラスの**「振る舞いサブタイプ」**であることが保証され**プラグイン可能なSW部品化**となる



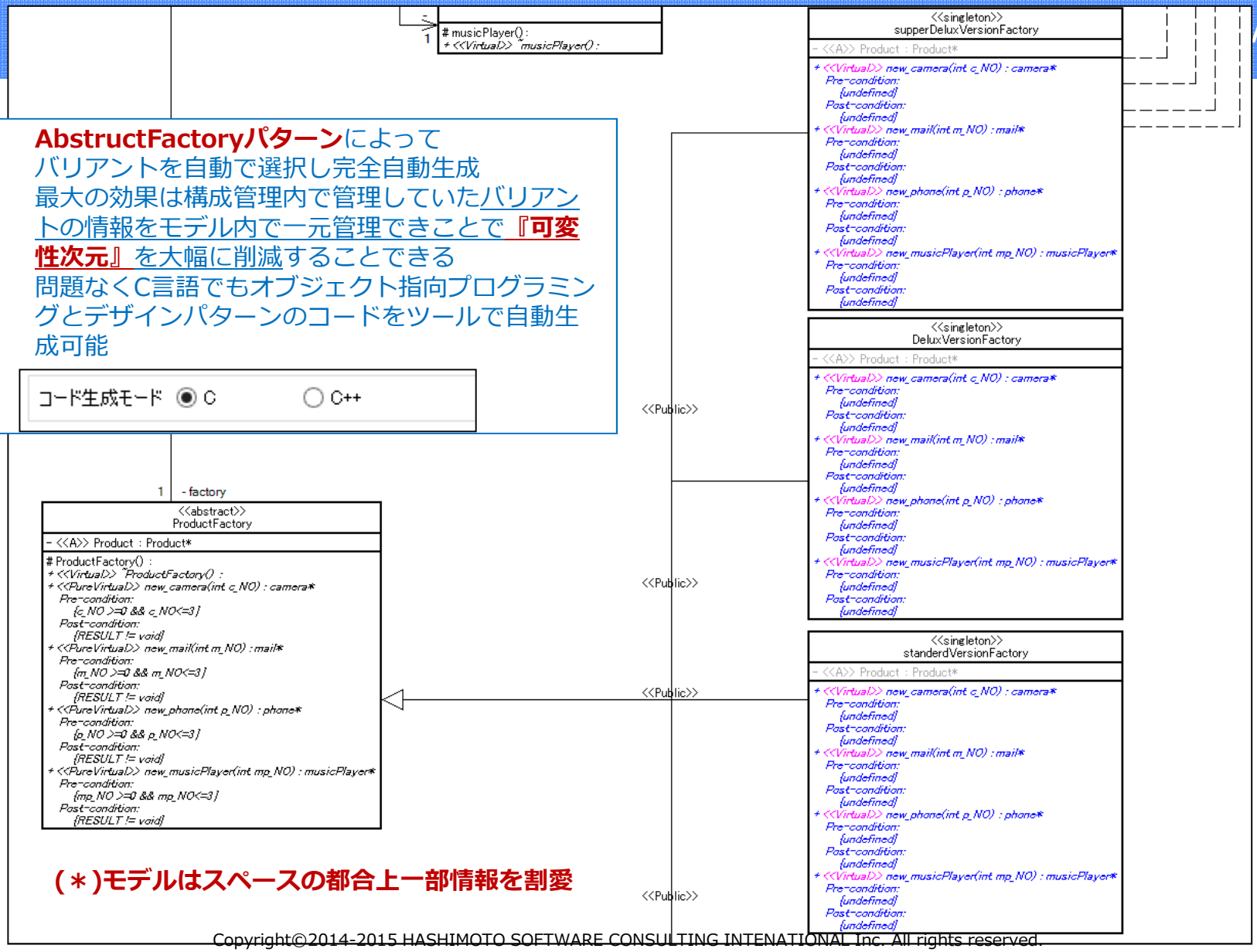
(＊)モデルはスペースの都合上一部情報を割愛

バリアントの自動デプロイメントの仕組み (モデルの続き)

HASHIMOTO SOFTWARE CONSULTING INTENATIONAL Inc.

- **AbstructFactoryパターン**によって バリアントを自動で選択し完全自動生成
- 最大の効果は構成管理内で管理していたバリアントの情報をモデル内で一元管理できことで**『可変性次元』**を大幅に削減することできる
- 問題なくC言語でもオブジェクト指向プログラミングとデザインパターンのコードをツールで自動生成可能

コード生成モード ◉ C ○ C++

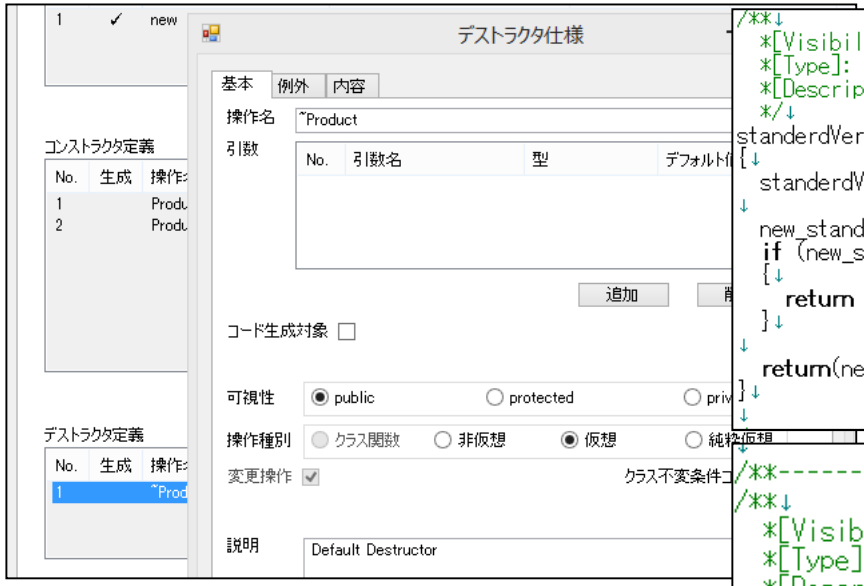


(＊)モデルはスペースの都合上一部情報を割愛

ツールによる自動コード生成～ C言語①

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

- C言語でもカプセル化／継承を含めたオブジェクト指向プログラミングとデザインパターンのコードをツールで完全自動生成が可能
  - 『開放閉鎖原理』
  - 『振る舞いサブタイプ』
  - 『契約による設計』のよる契約コードの生成とOn／Off切り替え
  - C++の純粋仮想／仮想／非仮想／静的関数の実装
  - C++の関数の可視性
  - C++のnew／コンストラクタ／デストラクタ
  - その他



newの実装例

```
/**↓
  *[Visibility]: Public↓
  *[Type]: Static↓
  *[Description]: Default Generator↓
  */↓
standerdVersionFactory* standerdVersionFactory__new(void)↓
{↓
  standerdVersionFactory* new_standerdVersionFactory;↓
↓
  new_standerdVersionFactory = (standerdVersionFactory*)malloc(sizeof(standerdVersionFactory));↓
  if (new_standerdVersionFactory == NULL)↓
  {↓
    return NULL;//例外処理を行うときは例外コードを指定する↓
  }↓
↓
  return(new_standerdVersionFactory);↓
}↓
↓
```

仮想デストラクタの実装例(の一部)

```
/**------------デストラクタインターフェイス定義-------------*/↓
/**↓
  *[Visibility]: Public↓
  *[Type]: Virtual↓
  *[Description]: Default Destructor↓
  *[Pre-Conditions]: ↓
  *[Post-Conditions]: ↓
  */↓
void cameraType#4__destruct(cameraType#4* const this)↓
{↓
  (this->method_table->destruct)(this);↓
}↓
```

ツールによる自動コード生成～ C言語②

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

クラスの実装例

```
↓
/**-----------------メンバー関数マクロ定義-----------------*/↓
#define PRODUCTFACTORY_MTHD_TBL¥↓
  void (*destruct)();¥↓
  camera* (*new_camera)();¥↓
  mail* (*new_mail)();¥↓
  phone* (*new_phone)();¥↓
  musicPlayer* (*new_musicPlayer)();↓
↓
/**-----------------メンバー関数構造体定義-----------------*/↓
typedef struct ProductFactoryMethodTable {↓
  PRODUCTFACTORY_MTHD_TBL↓
} PRODUCTFACTORY_METHOD_TABLE;↓
↓
/**-----------------クラス変数マクロ定義-----------------*/↓
#define PRODUCTFACTORY_CLASS_DT↓
↓
↓
/**-----------------メンバー変数マクロ定義-----------------*/↓
#define PRODUCTFACTORY_DATA¥↓
  Product* Product;↓
↓
/**-------------------クラス構造体定義----------------------*/↓
struct ProductFactory {↓
  PRODUCTFACTORY_METHOD_TABLE* method_table;↓
  PRODUCTFACTORY_DATA↓
};↓
↓
/**-------------------コンストラクタ宣言--------------------*/↓
void ProductFactory__construct(ProductFactory* const this);↓
```

仮想関数の実装例の一部

```
↓
/**↓
  *[Visibility]: Public↓
  *[Type]: Virtual↓
  *[Description]: ↓
  *[Pre-Conditions]: c_NO >=0 && c_NO<=3↓
  *[Post-Conditions]: RESULT != void↓
  */↓
camera* standerdVersionFactory__new_camera(standerdVersionFactory* const this, int c_NO)↓
{↓
  return (this->method_table->new_camera)(this, c_NO);↓
}↓
↓
/**↓
  *[Visibility]: Public↓
  *[Type]: Virtual↓
  *[Description]: ↓
  *[Pre-Conditions]: m_NO >=0 && m_NO<=3↓
  *[Post-Conditions]: RESULT != void↓
  */↓
mail* standerdVersionFactory__new_mail(standerdVersionFactory* const this, int m_NO)↓
{↓
  return (this->method_table->new_mail)(this, m_NO);↓
}↓
↓
/**↓
  *[Visibility]: Public↓
  *[Type]: Virtual↓
  *[Description]: ↓
  *[Pre-Conditions]: p_NO >=0 && p_NO<=3↓
  *[Post-Conditions]: RESULT != void↓
  */↓
phone* standerdVersionFactory__new_phone(standerdVersionFactory* const this, int p_NO)↓
{↓
  return (this->method_table->new_phone)(this, p_NO);↓
}↓
```

ツールによる自動コード生成～ C言語③

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

『契約による設計』の演繹推論コードの例

```
/**↓
  *[Visibility]: Public↓
  *[Type]: PureVirtual↓
  *[Description]: ↓
  *[Pre-Conditions]: mp_NO >=0 && mp_NO<=3↓
  *[Post-Conditions]: RESULT != void↓
  */↓
musicPlayer* ProductFactory__new_musicPlayer_(ProductFactory* const this, int mp_NO)↓
{↓
  /**----------------------事前条件--------------------------*/↓
  PRE_CONDITION("Violation of Pre Condition: ProductFactory.new_musicPlayer", (mp_NO >=0 && mp_NO<=3))↓
↓
～～↓
中略↓
～～↓
↓
  /**----------------------事後条件--------------------------*/↓
  POST_CONDITION("Violation of Post Condition: ProductFactory.new_musicPlayer", (RESULT != void))↓
↓
}↓
```

まとめ①

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

科学的モデリング規則：『科学的SW-Plug＆Play』と『科学的SW-バリアント・デベロップメント』を実践する

- **『科学的SW-Plug＆Play』**と**『科学的SW-バリアント・デベロップメント』**で課題解決
  - 『多品種少量生産』
  - 『アウトソーシング／オフショア』
  - 『分散並行開発』
- モデリングツール・構成管理による完全自動化
  - **演繹推論エンジン**で**モデルの正当性・妥当性を自動チェック**
  - **正当性・妥当性のあるモデルから自動コード生成**
  - 先進的構成管理技法による完全自動デプロイメントを実現
    - 欧米の制御系のソフトウエア開発企業は,1000以上のバリアントの選択と組合わせが可能でデプロイメントまでを**完全自動化**されている
- **オブジェクト指向技術**と**モデリング技法**および**構成管理**の連携が重要
  - 『タイプ置換原理』
  - 『開放閉鎖原理』
  - 『振る舞いサブタイプ』
  - 『Abstract Factoryパターン』

まとめ②

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

科学的モデリング規則：
ソフトウエア部品は『タイプ置換原理』で「振る舞いサブタイプ」にせよ

- サブクラスが満たすべき重要な要件はスーパークラスの**「振る舞いサブタイプ」**であること
- 『リスコフ置換原理』によるモジュラー推論で,継承階層やクラスライブラリのクラスのサブクラス／コンポーネントは**拡張に開いていることを保証**する
- 『リスコフ置換原理』によるモジュラー推論で,継承階層やクラスライブラリに新たなクラスのサブクラス／コンポーネントも**クライアントのコードが再コンパイルや再テストする必要が無いことを保証**する

＊『リスコフ置換原理』以外の置換原理でもモジュラー推論は可能

まとめ③

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

科学的モデリング規則：モジュラー推論(Modular Reasoning)で安全な派生開発と再利用を行うことを保証する

- 新しいサブクラスを追加したとき**意味的な整合性のチェック**をすること
- 予期しない振る舞いが無いかを**モジュラー推論で判定**をすること
- スーパークラスのオブジェクトを使用するクライアントコードが,継承階層内のサブクラスを利用しても**意味的整合性が保持できることを保証**をすること