

ご質問

ご回答

清水講師

1	変更仕様から元の要求を考えることは、ソフトウェアが大きくなると難しくなりますが適切なソフトウェア規模はどの程度でしょうか？	この場合の「元の要求」とは、この変更が発生した仕様を包含する機能という事であれば、規模の大きさには無関係かと思えます。一般に規模が大きくなると難しくなるというのはマネージメント技術の問題ですね。それでもここで規模が関係するとすれば、変更によって多くの機能の仕様に影響する可能性が高くなる点ですが、それでもXDDPによって、従来よりも対応しやすくなるはず（つまり前進するはず）です。また、一般に変更は「仕様」のレベルで届くということで、変更理由や背景も含めてこの変更仕様から「変更要求」を考えるのが難しくなるのでは、という意味でしたら、ソフトウェアの規模は無関係というお答えになります。
2	XDDPの書式はどの程度のソフトウェアサイズに適していますか？	「ソフトウェアサイズ」がベースの規模ということでしたら、規模が影響するのはマトリクスの「列」情報を扱いにくくなる、というところですが、現状ではXDDPに代わる方法は無いと思いますので、「工夫」していただくしかありません。ただ、規模が大きくなると担当者が増えますので、Excel で運用した場合には複数の担当者による同時アクセスに制限が生じます。その点では、今回日立IGSから公表されたツールは「セル」単位のロックになっていますので、この制約は解消されます。
3	要求が相互に依存しているとき、XDDPではどのように取り扱えばいいでしょうか？	依存関係にある機能に対する変更ということであれば、上位概念として変更要求を設定し、その中で依存関係にある機能の仕様変更を扱う事になります。逆に、一般の方法では、いくつかの変更が依存関係の中で変更された（されようとしている）仕様であることを表現できないでしょう。
4	サブシステム毎に変更要求仕様書を作成するとき、どのような注意が必要ですか？	変更要求仕様は、システム全体を扱うことをお勧めします。ただ、その場合 Excel では「列」情報の表現に制限が生じる可能性があり、自分以外のサブシステムをそれぞれ「1列」で扱う事で対応できるでしょう、適当なツールがあればこの種の問題は解決します。
5	変更要求が順次、追加されていく開発にXDDPは適用できますか？	変更にもなる影響がほとんど及ばないようなシステムであれば、そのままXDDPでも対応できますが、影響がいろいろな機能に及ぶようなシステムの場合、順次追加されてくる変更案件に対して、変更要求仕様書(TMを含む)のところで作業を止めておき、関連する変更案件をいくつかまとめて変更要求仕様を調整した上で、その先の作業に進める方法があります。それでも、あるべき姿としては、そうした発生形態そのものを改めるべきでしょうね。いずれ行き詰まる危険が高いですから、何でも解決する方法というものはありませんので。
6	レビューによるバグ検出率は、どれほど高くなりますか？	直接的な数字は持っていませんが、総じて、第3者テスト以降のバグの検出率は、それ以前の1/5~1/10に減ります。これはソースコード変更以前のレビューの効果、あるいは「3点セット」の成果物を作成する過程で担当者自身が気付いて訂正されていることを示しています。
7	XDDPでは、「一気にソースコードを変更」することになっているということですが、私の経験では一気にソースコードを変更するとかえって生産性が落ちるようになって思えます。アジャイルでは、少しずつソースを変更し、その都度テストすることが行われます。このやり方のほうが生産性が向上する様に思えますがいかがでしょうか？	ここでいう「生産性」というのは、実装工程におけるコード生産性のことです。XDDPでは、「3点セット」で変更箇所の準備(レビューを含む)を終えたあとで、一気にコードを変更しますのでこの生産性の値は高くなります。変更にもなる影響が生じにくいシステムでは、いわゆるアジャイルな方法で逐次対応して生産性が上がる(あるいは落ちない)かもしれませんが、影響が拡散する状況では、その都度ソースコードを変更したのでは変更のし直しが生じます。組み込みシステムでは、新しいH/Wの準備に時間がかかることもあって、まとめた変更になることが多いのです。

8	<p>PFDを設計部署に紹介すると、作業順の一本道で示して欲しいというニーズが出てきます。PFDの誤解につながるのではと危惧しますが、意味はあるのでしょうか？</p>	<p>プロセスは、本来実際に作業をする人が作成する(設計する)ものです。そうでないと、途中の状況の変化に対応できません。実際、途中で要求が追加されたり、初期の見積り甘さが表面化したりして、スケジュールが遅れてきます。そのような場面においては、当初考えた開発アプローチに対して「別案」を考え出す必要があります。PFDの狙いは、「シミュレーション」によって、変化させたプロセスを安定させる事と、状況の変化に対して「別案」の開発アプローチを考え出すことにあります。「一本道」というのはいわゆる「ワークフロー」ですね。その場合の最大の障害は「別案」を考え出せないのではないかと思います。変化する「QCD」に応え続けるには、PFDのようなプロセスを自在に設計するツールが不可欠です。</p>
9	<p>変更設計書にアクティビティ図やフローチャートなどを書くことは問題でしょうか？</p>	<p>まず第一に、XDDPで作成している「3点セット」の成果物は、今回の変更作業の「差分」情報であり、「メモ」的な情報に過ぎないということです。最後にこれらの「差分」情報が公式の成果物に反映(マージ)されれば役目は終わります。その時点でこれらの成果物自身は「変更記録」として残るだけで、いわゆる設計書のような公式の成果物として残る物ではありません。アクティビティ図やフローチャートというのは、現状のソースコードを解析した結果の物ですので、いわゆる「スペックアウト資料」として扱えば良いでしょう。変更設計書にはそうして把握した状況に対して変化点だけを簡潔に書きます。この過程で作成したスペックアウト資料は基本的には公式の成果物として残される物ですので、変更があったときは、終了後に今回の変更設計書の内容がスペックアウト資料に対してマージされます。</p>
10	<p>V字の左側」を確実に実施してバグを減らし、テスト工数が減るとの考えだと思いますが、テストそのものの工夫(やり方)として工数減とする方法はあるのでしょうか？ 例えば、変更箇所が明確になりテスト項目は減るのでしょうか？ 論理的には難しいと思っています。レビュー次第でしょうか？</p>	<p>もともと派生開発では、既にテストされて稼働しているシステムを変更しますので、バグは、今回の変更に関係した箇所や内容にあることは明らかです。XDDPでは、具体的なレベルで変更箇所が記述されることで、いわゆる変更方法まで明らかになってレビューされていますので、設計部門とテスト部門が連携する事でテストの効率を上げることができるはずですが、現状の問題は、これらの組織間の連携ができていない為に、テストが過剰になっていると思っています。もちろん、今回の変更箇所とは別に、品質保証の観点からのテストは最小限必要です。</p>
11	<p>XDDPが適用できる範囲がわからない。」何が制約になり得るのか？ どのような場合に適用が難しいのか？ (100%適用できるということはありません)</p>	<p>適用できない領域はないと思っています。あるとすれば、適用のしやすさや適応の効果におけるばらつきでしょう。それでもできない事を証明する意味はありませんので、どうすれば「そこ」でうまく適用できるかを工夫したほうが良いと思います。制約は、製品やシステムの領域にあるのではなく、組織の習慣やルールにあると思っています。</p>
12	<p>USDMの記法はある程度要求がかたまっているのが可能だと思いが、例えば**アルゴリズムを入れるだとか、プロトを作らないとわからないものはどうするのか？</p>	<p>昔の「紙」の時代と違って、今の時代は「仕様は書きながら決めていく」あるいは「決めながら決めていく」ことができます。途中で、そこまで前提としてきた仕様が行き詰まった時は、必要なところに戻って修正すれば済みます。いわゆる試作が必要な箇所も、そうして仕様化した過程で把握できますので、あとでまとめてプロトタイプングでも何でも対応できます。プロトを作るにしても、関連する仕様は「暫定」として決めていく方が効果的です。</p>
13	<p>TMが巨大(ファイル、関数が多すぎるとか)すぎて、作成のコストがでかい場合どうするのか？</p>	<p>関係するファイルの数が多くなるとTMは巨大になります(関数の数は無関係)。でもTMは一度作ればそれ以降は殆ど再利用できますので、1、2回の派生開発で工数は回収できます。それに、今の時代はファイル情報を集めるのはそれほど難しい作業ではありません。もしかするとほとんど「コピー&ペースト」でできるかもしれません。</p>
14	<p>テスト後にベース文書にマージすると、トレーサビリティを要求する法規格に引かかる場合があるが？</p>	<p>そこでマージする「差分情報」は、すべて公式にレビューされ、それに基づいて変更されたソースコードが第三者によってテストされていますので、それらの差分情報は正しい(少なくとも適切である)と判定されたこととなります。公式の文書に対して、これらの差分情報に基づいてきちんとした構成管理下でマージすることのほうが理にかなっているはずですが。</p>

15	開発工程が「システム仕様①」「仕様②」「設計③」「コーディング④」とあった場合、TMは誰がどこで作成するものでしょうか。個人的には②と③の間かと思いますが不明点としては誰が作るかという点です。②と③は別の担当者、別会社である可能性があり、成果物として依頼ある必要があるのかと思います。②と③のどちらでもいいのか、理想はどちらかなのか、理由も含めてご教授ください。	変更プロセスの中でTM(変更TM)を作成する工程が①～④のいずれか?というご質問であれば、これらの工程(①～④)は新規開発の工程であって、XDDPの「変更プロセス」において想定するものとは異なります、というお答えになります。XDDPでは、変更仕様を抽出する作業の中で変更TMを作成しますが、機能仕様書を見ながら変更仕様を抽出することもありますので、その場合は関数特定できていませんので、せいぜい変更が予想されるソースファイルに「O」が付く程度です。関数名が入るのは、ソースコードを見るプロセスの中です。 本来、TMは新規開発(あるいは派生開発の追加機能の開発)の中で、②や③(最終的に関数レベルの設計)で、設計作業を進めながら作成します。そこで仕様に対してTMが作成されていれば、派生開発の場面では、変更仕様を抽出する中で変更仕様と一緒にTMの情報を抽出して「変更TM」に転記するだけです。
16	XDDPでは、どのようなタイミングで、どのような見積もりを行うのでしょうか?代表的な例で教えていただきたいです。	今回の変更要求にマッチする開発アプローチをPFDを使って「成果物とプロセスの連鎖」で表現します。そしてこの連鎖を使って成果物のサイズを見積もり、その成果物に作成するプロセスの工数を算出します(これが私の見積もり方法です)。最初の変更要求のレベルで、仕様上の変更箇所やソースコードの変更行数を見積もり、そこからソースコードの実装に必要な工数を見積もり(算出し)ます。この工数は担保することになりますので、作業を進める中で、常に変更行数の見積もり値を監視することになります。「Software People vol.5」があれば参考にしてください。
17	XDDPでは、どれくらい小さい規模(工数)までを想定されているのでしょうか?	規模の大きさで対応を変えることは想定していません。数100行の1人プロジェクトでも有効です。組織のメンバーがレビューに参加することで孤立することも回避できます。逆に、「1人プロジェクト」で対応できれば、多人数のプロジェクトでもうまくいくはずですよ。
18	モジュール間のソースコードの影響範囲を調べていくのに、工夫して効率を高めている事例がありますか?	変更に対する影響には①機能仕様のレベルで関連するケース、②設計(実現)の方法によって関連するケース、の2通りがあります。「3点セット」の成果物は主に①のケースに有効ですが、②に対しては、設計書やその場で作成するスペックアウト資料が機能します。それでも十分ではありませんので、これらの相関を表現するための「補助資料」を用意しておくことをお勧めします。
19	TMレビュー時に、ソースコードをしらべていないレビュワーが見るためのポイントがありますか?	その箇所のソースコードにもっとも詳しいのは担当者です。その状況でレビューアの参加が効果を上げるには、彼の経験あるいはソフトウェアの設計の常識やパターンなどが活かせる状況を作る事です。たとえばTMにはソースファイル名の情報以外に、そのソースコードがカバーする機能などの単位でまとめたり、左から右に時間軸で並べたりします。すくなくともその並びはいつも同じである事です。
20	USDM/XDDPで非機能要求はどう扱いますか?組込みの現場では機能のみに注目するくらいを感じているので	一般に、組み込みシステムの従事者は「ボタンだらけ」の製品を作るのが得意で、これまでは操作性などの品質要求は重要視されてきませんでした。最大の理由は組織としてソフトウェアエンジニアリングの習得を怠ったことでしょう。もう一つの原因は、機能仕様書と要求仕様書の違いを認識していないことです。機能仕様書の書き方では「保守性」のような作り方に関する品質要求(の仕様)を表現できません。 もともと非機能要求という表現は間違いを起こしやすいと思っています。「ユーザーの要求に非ず」という意味に使われることもあるようです。作業上の制約に関する要求も含まれることがありますが、一般には品質要求のことを指すと理解しています。 品質に関する非機能要求で、たとえば「操作性」において「画像のスムースな動き」が求められた場合、関係する機能の仕様や実現方法に影響を与えたり、制限を加えることもあります。USDMでは、必要な品質特性に対して通常は「要求(変更要求)」を設定する事を勧めています。機能に付随する品質の中には、特定の機能の仕様レベルで対応するケースがあり、その場合は仕様のグループまたは2層目の要求に品質特性を設定し、その品質特性に応じた仕様を記述します。ただ、機能要求と違って、品質特性によっては実現方法に触れないと記述できないことがあります。
21	XDDPとウォーターフォールやインクリメンタル等の開発プロセスとの関係はどうとらえたら良いのか、ご教示ください	XDDPには2つの開発アプローチを含んでいて、変更側の開発アプローチはいずれの定義にも当てはまりません。それでも「3点セット」の成果物が少なくとも2つの段階に分かれて作られることに着目すればウォーターフォールに見えるでしょうし、必要なことだけを実施しているところに着目すればアジャイルに見えるでしょう。既存の型に当てはめる必要はないと思っています。一方、追加側に関しては、新規開発のアプローチのどれでも対応できます。

22	汎用コンピュータの開発プロセスでは実施されていたが、XDDPでは省略したものはありますか？ あれば例を紹介してください	追加側の開発アプローチは、基本的にオーソドックスなアプローチですので省かれているものは無いはずですが、変更側の開発アプローチでは、いわゆる「差分開発」ですので、まったく異なっています。
23	before/afterを一行にしていますが、二行にすると正しく書けると思われます。また、before/afterに加え、相違点を書く、さらにレビューがしやすくなると思われれます。このようにすると書く人の負担が増えたり、表が大きくなってしまおうというデメリットもあると思いますが、ご意見をお願いします。事例は？	実際、1つの文章で「before」と「after」を上手に表現しにくい事もあり、その場合には before : after : と書くことを勧めています。ただ、変化点の前後に同じ文言を繰り返されると変化点が見えにくくなりますので、その場合は文章の途中で、変化する部分だけを before : after : と書く事もお勧めします。こうすれば、そんなに表が大きくなると思います。
24	ソフトウェアだけで成立する設計の場合は、初期段階でじっくりと時間がかけられればすんなりと導入ができそうな気がします。ただ、ハードウェア、メカなどが密に関係するような商品設計の場合、初期設計段階でのソフトウェアの提供を求められるケースが多々あります。このような場合 時間とコストに負担をかけずに対応するにはどのようなやり方が考えられるか？	これは現場では良くある話ですが、製品用のソフトウェアシステムを使ってハードの検査に使うの間に合わせるというのは無茶な話ですし、明らかにいくつかの点で合理性を欠きます。 ①ハードの人たちの使用(テスト)に間に合わせるためには、拙速な対応にならざるを得ませんので、製品のソフトウェアシステムとしての品質を作り込む工数が不足します。この種の品質は、後で盛り込みことは技術的にも時間的にも殆ど無理です。そのようなソフトウェアを製品として用いる事に対して誰が責任を負うのでしょうか。 ②製品用のソフトウェアでは、一般にハードに意図的に負荷をかけたたりする機能が含まれていません。つまり、これでは「とりあえず動く」としか確認できませんが、これでハードの設計者のプライドが許すのでしょうか。 という事で、本来は製品のソフトウェアとは別に用意すべきと考えています。それが競争力のある製品を開発する体制でしょう。
25	プロダクトライン要求分析における要求仕様の書き方 機能要求と非機能要求(作り方に関する要求)のどちらで対応するのがよいのか？ ①機能要求で対応した場合 要求仕様レベルで全製品に適用する仕様と特定製品に適用する仕様が出てくる。 本来、含まれる仕様の範囲が異なれば要求の表現が異なる。同じ要求の表現にも関わらず、仕様が異なるのは、漏れや抜けが生じやすくないか？それともコア資産とそれ以外で要求を分けた方がよいのか？あるいはくりうまく使うと対応できるか？ ②非機能要求で対応した場合 SPLに於ける設計は「変更する部分を変更できるように作る」ということで非機能要求として記述できるはず。しかし、可変性そのものは特定のフィーチャー、つまり機能に関する要求仕様として認識されるので非機能要求のみで記述することはできなさそう 上記①②の対応のうちどちらへんをおとし所にすればよいのか？	質問の意図が読み取りにくいのですが、複数の質問が混じっているようなのと、変更の質問ではないと判断して以下にお答えします。 ・非機能要求を「品質要求」とすると、機能に付随する品質要求と「保守性」のような作り方に関する品質要求があります。コア資産やアプリなどの通常の機能要求と、それらの作り方に関する品質要求はお互いに独立していますので、どちらかで対応するというものではありません。「保守性」で定義された具体的な作り方の仕様(この場合は、作業員に対する指示になる)を満たしながら、機能要求で求められている仕様を実現することになります。 ・コア資産とアプリとで要求を分けて扱っても構いません。ちょうど、コア資産の部分は共通機能のような位置づけにもなります。その場合、相互の関連を何らかの方法で明らかにする必要があります。これに対して、アプリの立場から一体で扱い、その中でコア資産に対して<<レイヤー>>という発想を持ち込んで表現する事もできます。この場合、アプリは「シナリオ」を扱う事になりますので、複数の経路で同じコア資産に繋がることが起きるので記述ルールの確立が必要です。

26	コード生産性+要求仕様書等(3点セット)を作成する工数で見るとトータルの生産性はどれくらいになるのでしょうか？	異なる成果物を足してトータルの生産性というのは、成果物によって尺度が異なりますので算出できません。ご質問の意図が、新たに「3点セット」の成果物を生成した状態でトータルの工数がどれくらい増えるのかという事でしたら、基本的には増えません、というお答えになります。新しいことに対する不慣れな状態が、工数が増える方向に作用しますが、これらの成果物を書き留める工数そのものは、個々の変更箇所の探索過程や変更方法を考える中ではほんの僅かな時間です。それに対して、従来は、他の変更要求との関係で、既に変更したソースコードの箇所を何度も読み返していますので、書き留める工数は直ぐに回収できます。
27	XDDPを導入し、1回目から効果があるというホームページを拝見しましたが、初期導入コスト(教育等)はどの程度になるか。目安や試算方法はありますか？	必要な知識の習得と新しいやり方に対する練習が必要ですが、ある程度、事前に準備されていれば直ぐに効果が得られます。具体的には、一般のソフトウェアエンジニアリングにおけるプロジェクト計画書を構成する項目に関する一連の知識が必要で、これの不足に応じて習得が必要です。この領域では「知っている」ことで対応できる項目が幾つかあります。その上で、PFDによるプロセスの設計やUSDM、XDDPの準備が必要ですが、これらは習慣化が必要です。一般のソフトウェアエンジニアリングの中には含まれていませんので、別に対応することになります。現状の仕事をこなしながらということであれば2ヶ月ぐらい必要になるかもしれません。

中西講師

28	「予算」がコア資産(再利用対象?)という意味がよくわかりませんでした。もう少し説明いただけないでしょうか？	プロダクトライン間で共有される開発コスト計算の基礎となる諸表がコア資産になり得ると考えていただければよいと思います。
29	成功例をもう少し詳しく聞きたいです。	<p>きりがありませんので、文献の紹介をもって成功例の紹介とさせていただきます。</p> <p>1) SPL Hall of Fame, http://www.splc.net/fame.html; SPLCで選ばれた「殿堂入り」の事例集。日本企業では東芝の事例が取り上げられています。</p> <p>2) Paul Clements and Linda Northrop, Software Product Lines: Practices and Patterns, Addison-Wesley, 2000; 本の後のほうに事例集。ただ、現在となつては古い事例だと思います。</p> <p>3) Frank J. van der Linden, Klaus Schmid, and Eelco Rommes, "Software Product Lines in Action," Springer, 2007; 事例が豊富。</p> <p>4) 日立製作所と日立ハイテクノロジーズの事例: 製品単位でプロジェクトが組まれる会社でのSPL導入事例。Yasuaki Takebe, Naohiko Fukaya, Masaki Chikahisa, Toshihide Hanawa, and Osamu Shirai, "Experiences with Software Product Line Engineering in Product Development Oriented Organization," Proc. Software Product Line Conf. 2009, pp.275-283, Sep. 2009.</p> <p>5) 富士通九州ネットワークテクノロジーズの事例: SPLの全社的導入事例。5.1) Takashi Iwasaki, Makoto Uchiba, Jun Ohtsuka, Koji Hachiya, Tsuneo Nakanishi, Kenji Hisazumi and Akira Fukuda, "An Experience Report of Introducing Product Line Engineering across the Board," Proc. 14th Int. Software Product Line Conference (SPLC) 2010, Vol.2, pp.255-258, Sep. 2010.</p> <p>5.2) 岩崎 孝司, 内場 誠, 大塚 潤, 八谷 浩二, 中西 恒夫, 久住 憲嗣, 福田 晃, 『プロダクトライン開発手法の全社的導入に関する一事例報告』, 組込みシステムシンポジウム2010(ESS2010), 2010年10月。</p>
30	コア資産の進化にXDDPが適用できませんか？	適用できると思います(同意します)。ただ、プロダクトラインの全体像を無視したコア資産の「派生開発」はSPLのうまみを減却してしまうと思います。フィーチャモデル、さらにはアーキテクチャを横目で見た上でのXDDP適用が必要だと思います。また、その際に作成される変更要求仕様書、トレーサビリティマトリックス、変更設計書はフィーチャの追加/変更、さらにはフィーチャモデル変更の基礎資料として利用可能であると考えます。
31	P-39で機能要求は言及されていますが、非機能要求はどう扱いますか？特にアーキテクチャ定義(設計)では重要な入力となると思います。	<p>非機能フィーチャから非機能要求実現手段(モノ、コト)への紐づけをどう実現するかの問題になると思います。以下のような方法が考えられると思います。</p> <p>1) 非機能要求の機能要求への変換: たとえばQFD等を用いて、非機能要求を実現する機能をそのインパクトとともに見える化する。</p> <p>2) 非機能要求のプロセスへの変換: 開発プロセスをPFDで見える化し、非機能要求から非機能要求を実現するためのプロセスをそのインパクトとともに見える化する。</p> <p>3) リアルタイム性についてはタスク構成やリアルタイムスケジューリングのひな形、メモリ消費量/消費電力についてはメモリ消費量/消費電力見積りのためのモデルをつくってプロダクトラインで共有する。</p>

32	SPLとXDDPの共通性、相違性にご教示ください。また、その共通性、違いが何故生じているのかについても教えてください。	<p>パネルでも述べましたが、私は両者の相違性について以下のように考えています。</p> <ol style="list-style-type: none"> 1) SPLはパラダイム、XDDPは方法論 2) SPLは計画駆動、XDDPは変更駆動 3) SPLは全体理解志向、XDDPは部分理解容認 4) SPLはアーキテクチャ志向、XDDPはアーキテクチャはなくてもよい 5) SPLはいきなり変化点の実現に飛びつかない(変化点を可能な限り出し尽くす)、XDDPはいきなりコードの変更に飛びつかない <p>これらの相違点の理由ですが、SPLは予測あるいは計画指向であるのに対し、XDDPは変更駆動であるところが一番大きいと思います(上の箇条書きとかぶっていますが…)。</p> <p>両者の共通点は、派生製品開発で生じる「変更のモグラたたき」を避けることだと思います。人間の理解を目的とし人間が把握できる分量にまとめられる上流資産と異なり、コードは人間の思考からかけはなれ、かつ一人の人間では把握できない分量の資産となっています。こうしたものを部分的にアドホックに書き換えると、後でどこでどうなるかわからない。人間が把握できる上流資産で変更を計画するのが両者の共通点だと思います。</p>
33	SPLが向いている製品、XDDPが向いている製品の見極めの方法はあるのでしょうか？	<p>少なくとも導入障壁の面ではXDDPの方が有利ではないかと思えます。定量的な根拠がある訳ではないですが、私見としては、先の製品がある程度見通せてコア資産を固められるのならSPL、見通せないのならXDDPがよいと思います。(せっかく作ったコア資産が無駄になるリスクを避けたい。たとえば、今後どんな製品が出るか想像もつかない「情報家電」のコア資産なんて決められるでしょうか？)</p>
34	時計ならわかるが、一般的にfeatureは機能なのか、Caseなのか？つまり、ユースケース図での丸にあたる部分の事をSPLのフィーチャーとしているのか？	<p>フィーチャの定義はいろいろあるのですが、フィーチャモデルの提唱者による定義は「システムのさまざまなステークホルダによって認識可能なシステムの特徴」です。ですので、フィーチャは識別可能な機能の場合もありますし、機能でない場合もあります。</p> <p>つぎにフィーチャとユースケースの関係について述べます。ユースケースで記述できるのは基本的には機能要求のみですから、上記のフィーチャの定義からもわかるように、フィーチャはユースケースに対応づけられますが、フィーチャが対応づけられるのはユースケースだけではありません。</p> <p>例を挙げます。時計の例において、「目覚ましを鳴らす」、「目覚まし時刻を設定する」というユースケースがあった場合、いずれも「目覚まし機能」に関するものですから、これらふたつのユースケースを「目覚まし機能」というフィーチャにマッピングすることが可能です。あるいは、モデリング実施者によって、「目覚ましを鳴らす」を「目覚まし鳴動機能」というフィーチャに、「目覚まし時刻を設定する」を「目覚まし時刻設定」というフィーチャにマッピングするかもしれません。このようにどのようなフィーチャを定義するかはモデリング実施者に依存します。目覚ましを鳴らす、目覚まし時刻を設定する機能は普通どの目覚まし時計にもありますからほとんどの人にとっては前者のモデリングが自然でしょうが、もし万が一目覚まし時刻を設定できない機種がプロダクトライン中にあるなら後者のモデリングをとるべきでしょう。</p> <p>また、複数のユースケースを横断的にまたがるようなフィーチャも考えられます。「目覚ましを鳴らす」、「時報を鳴らす」というユースケースがあり、これらのユースケースでは機種によって日本語、英語、中国語のいずれかで目覚まし音声、時報音声を再生するものとします。この場合、「日本語」、「英語」、「中国語」のそれぞれがalternativeなフィーチャとなりますが、これらのフィーチャはユースケース(の一部の記述)を横断的にまたがっており、ユースケース全体の抜き差しができる形にはなっていません。さらに「日本語」、「英語」、「中国語」といったフィーチャは機能を表すものではないこともわかりたいだけだと思います。</p> <p>こうしたフィーチャとユースケースの関係については、Hassan Gomaa, Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architecture, Addison-Wesley, 2004. によくまとめられています。</p> <p>フィーチャは要求レベルの相違のみを表現する抽象概念ではありません。要求より低い抽象度の相違を表すことがあります。たとえば、時計の時刻更新のために、時計LSIに「ポーリング」をかけるか、あるいはタイマLSIから「割込み」をかけるか、製品によって設計が異なるとしたなら、「ポーリングによる時刻更新」、「割込みによる時刻更新」といったフィーチャを定義しなければなりません。これらのフィーチャはすでに要求レベルのものではなく、設計レベルのものであることもご理解いただきたいと思います。</p> <p>フィーチャは非機能要求の相違を表すことすらあります。たとえば、時計の電池の持ち時間が製品によって「1年」、「3年」と異なるなら、これらもフィーチャとして定義することになります。</p> <p>乱暴ではありますが、まとめると、フィーチャとは「製品間で異なる何か」です。</p>

35	各コンポーネント、モデルにおけるコンフリクションはどう評価しているか？	申し訳ありませんが、事例を知りません。
36	一番大事なのは、Business 特に要求の獲得が洗い出しが大切だが、お客から要求されていない要求をつくるのは一番大変だと思う。ここでのSPLで方法論などはあるのか？ 普通はお客からの要求を獲得し分析していけばよいが	<p>要求の獲得の大切さ、難しさはSPLに限った話ではなく、SPLの範疇外の話ではないかと思います。つまり既存の要求獲得技術を用いて要求を獲得していくことになると思います。</p> <p>SPL的に大事なのは獲得された要求間の「違い」を整理することになります。フィーチャモデリングを用いて、すなわち「違い」の分解、「違い」の抽象化、何かしらの観点／概念軸に基づく「違い」の整理の過程で、自身が気づいていなかった要求やお客から（現在は求められないが）将来求められそうな要求が見つかる機会が増えることでしょう。また、想定していなかったお客さんの要求がそれらの「違い」の組み合わせで実現できるようになる可能性も増します。</p> <p>「違い」を適切な粒度まで分解し、「違い」を容易に実現できる仕組みを作ることで、後からの大幅なソフトウェア変更のリスクを下げるのがSPLの真髄だと思います。</p>
37	XDDPの説明の中で「TMが書けないとSPLに影響する」というお話がありました。TM3点セットとSPLでのドキュメントについては、どのような関係で考えられるでしょうか？	<p>XDDPは部分理解容認型の方法論であり、変更要求からのトレーサビリティ確保を重要視しています。一方、SPLは全体理解追求型の方法論であり、「可変性」からのトレーサビリティ確保を重要視しています。いずれもトレーサビリティ記述にTM等を使用することになりますので、「XDDPのTMが書けないならSPLは無理」というのはその通りだと思います。</p> <p>SPLは方法論というよりパラダイムですので、方法論によって多少異なりますが、講演でお話したフィーチャ指向SPLでは、（変更要求ではなく）フィーチャから各種ドキュメントへのトレーサビリティ確保を図ります。この各種ドキュメントには、追加要求仕様、変更要求仕様も含まれます。すなわち、あるフィーチャからそのフィーチャに関係する追加要求、変更要求へTM等を介しての張られることとなります。TMの縦軸にはフィーチャ名、横軸には追加要求、変更要求、そしてXDDPでTMの横軸に書いていたものすべてが並ぶこととなります。</p> <p>追加要求、変更要求はフィーチャ発見の源泉ともなるでしょう。ひとつひとつの製品で同時に実現されるひとつ以上の追加要求、変更要求をパッケージし、わかりやすく表現する名前を見つければそれがフィーチャになり得ます。</p>
38	アプリケーションエンジニアリングとXDDPについて、もしかすると共通性があるかなと思いました。あるとしたらどのような点でしょうか？	<p>SPLのアプリケーションエンジニアリングは事前に予測され備えられた変更に対し、コア資産を事前に決めた方法で再利用して製品を作る計画駆動のアプローチであるのに対し、XDDPは事前に予測されていない変更に対し、変更管理を徹底して既存製品を作り替える変更駆動のアプローチであり、両者の出発点は大きく異なっていると思います。</p> <p>両者の共通点は、両者とも要求に始まりコーディング直前までの上流工程を重視し、トレーサビリティ管理を徹底するところだと思います。もっとも、SPLはドメインエンジニアリングにおけるトレーサビリティ確保を目指しますが（やはりSPLは全体理解指向）、XDDPは変更が生じてから変更に対する部分のみのトレーサビリティ確保を目指すところで異なります（XDDPは部分理解容認）。</p>
39	SPLを支援するツール 環境の完成度 実用度について	<p>SPLのツールの役割は、フィーチャモデリングを含む可変性モデリングの支援、トレーサビリティ管理あたりにあるかと思いますが、SPL自体はパラダイムであり、方法論は多種多様、組織依存、あるいはプロジェクト依存になってくると思いますので、なかなか汎用的なSPL支援ツールは難しいかもしれません。</p> <p>商用ツールとしては、pure::variants、Gears、ZIPC Featureなどがあります。これらは実用に耐えるものとは思いますが（私はGearsは使ったことがないのでコメントできません）、可変性モデリングなどSPLの一側面をカバーするものです。SPLの開発活動全体を支えるものではないので、他のツールと併用したり、あるいは自社の開発プロセスにあわせてカスタマイズしてもらう必要があるかもしれません。</p> <p>研究ベースのものでしたら多くのツールが作られています、企業の製品開発における実用には耐えるものはほとんどないと思います。ツール自体がbuggyであったり、大規模な開発に耐えなかったり、特定の метод論に依存してあたり、あるいは使用契約で実開発での使用を禁じているケースもあります。（あくまで研究のためのプロトタイプ、あるいは研究評価用ですから。）</p>
40	コア資産の進化にXDDPが適用できませんか？ 線引きの事例	上の林氏への回答をもって回答に変えさせていただきます。

41	フィーチャーツリー作成のガイドラインにはどんなものがありますか	<p>下記の論文でガイドラインが紹介されています。 Kwanwoo Lee, K. C. Kang, and Jaejoon Lee, "Concepts and Guidelines of Feature Modeling for Product Line Software Engineering," Proc. of the 7th Int. Conf. on Software Reuse (IGSR), pp.62-77, 2002. また、手前味噌で申し訳ありませんが、私の方でもガイドラインを執筆しております。(ご批判もあるガイドラインであることにもご注意ください。)</p> <p>中西恒夫, 久住憲嗣, 福田晃, 「ソフトウェアアーキテクチャ事前設計を目的とするフィーチャモデルのガイドラインとアンチパターン」, 信学技報, Vol.109, No.170, SS2009-25, pp.78-82, 2009年8月. 中西恒夫, 久住憲嗣, 福田晃, 「ソフトウェアアーキテクチャ事前設計を目的とするフィーチャモデリングのアンチパターン」, 組み込みシステムシンポジウム (ESS2009) 予稿集, 2009年10月. またガイドラインを定める以前に、何のためにフィーチャーツリーを書くのか、そのスタンスを明確にしておくことも重要です。製品のちがいの見える化を主目的にするのか、それともソフトウェアのレイヤ構造を粗く決めるためにするのかでも、描き方が変わってきます。</p>
42	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体制比率で理想的なバランスはありますか？	<p>アプリケーションエンジニアリングはごくわずかの人間で行われ残りの大部分はドメインエンジニアリングに回っている、そして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両方に携わっている人員が従来開発の場合より少なくなっている…というのがSPLの教科書的理想です。しかしながら、大多数の会社さんは段階的にSPLを導入しかないと考えます。初期のうちはコミュニケーションのよい少数のエキスパートチームがSPLの計画とコア資産の開発に従事し、残りの大多数はコア資産の再利用と非SPL部分の開発に従事するほうが現実的だと思います。その後、SPLが社内に浸透するにつれて、ドメインエンジニアリングを増強していくとよいと思います。実際、講演で紹介した富士通九州ネットワークテクノロジーズの事例では、プロジェクト自体は300人超でしたが、ドメインエンジニアリングチームは数人でした。</p>

桜庭講師

43	進化型SPL XDDPではやはり部分理解であり、構造が明解であってもSPLのコンポーネントとして使えるのだろうか？ 言い換えればXDDPの修正資産はSPLのコア資産の条件に足るのか、または、足るためには何がポイントか？	XDDPの修正資産をSPLのコア資産として採用するためには、既存のプログラムを改造する際に考慮すべきポイントがいくつかあると考えています。将来にわたっての共通性や共通部品としての品質や保守性などについて設計・テスト時に考慮すべきだと考えます。(単純な改造レベルではコア資産としては不十分なケースが多い) 進化型SPLでは最初に製品分析プロセスがあり、全体のフィーチャ分析をしていて、ここでコア資産としての要件を検討することとしています。
44	P-66についてXDDP導入したことで、テスト工数が減少した要因をご教授ください。	XDDPによりテスト着手前の品質向上が実現でき、その結果、テストフェーズでの不具合摘出が減って手戻り作業が減少した結果だと考えています。テスト工数というよりテストフェーズ工数が減少したと言うほうが正しいかもしれません。
45	SAGEPRO/eXMツールの開発規模を教えてください	既に開発していたプロジェクト管理ツールをベースに改造して開発しましたが、規模的には約30KSLOC程度です。
46	SAGEPRO/eXMのSAGEPROとは何の意味でしょうか	Sageとは「賢い人」の意味です。それにプロフェッショナルの「Pro」を付加しました。セイジプロと呼んでください。(サゲプロではありません(笑))
47	P-18にて、XDDPを適用して顧客先での不良がゼロではないのですが、いったいどのような内容(不良種別)だったのでしょうか？	改造仕様⇒コーディングの段階で不具合を作りこんでしまい、それをテスト前に摘出できませんでした。変更設計書のレビューが不十分だったと分析しています。
48	清水さんのお話の中で精度のよいプロジェクト計画書が必要との話がありました。試行と適用のそれぞれでどのようなものを作成したか教えてください。	もともと弊社ではプロジェクト計画時に作成する計画書がテンプレート化されており、今回はそのまま使用しています。(特にXDDP適用に際して従来と異なる計画書を作成していません)
49	進化型SPLのイメージがつきにくいのですが？追加機能をコア資産にしていくんですか？それとも母体も含めてリファクタリングするのですか？前者だとすると、アーキテクチャがあまりかわらないような気がしますが？	母体(改造箇所の周辺)も含めて部分的にリファクタリングしていくイメージです。このため、事前にゴールイメージの構造(アーキテクチャ)を検討し、文書化しておきます。
50	変更TM、変更設計書のみでは構造を明確にすることは困難であると思えます。この点についての工夫は？	改造母体のソフトウェア構造が不明確な場合は、市販の構造解析ツールやクローンコード検出ツールなどを使用して、改造箇所特定の参考にします。
51	変更設計書の数は多いと思います。どの様に管理していますか？	発表で説明したXDDP支援ツールSagePro/eXMを使って、多数の変更設計書を一元管理しています。
52	基本設計部へのマージは難しいと思います。工夫していることは？	基本設計部の意味を改造母体と解釈して回答します。 改造はXDDPで漏れなく正確に行い、的確な構成管理とリグレッションテストによる従来品質の保証になると考えています。
53	変更要求にひもつけてテスト項目を作成されているようですが意図しない変更がなされていないかのテストはどのように実施されていますか？考え方、工夫があれば教えてください	テスト項目として「意図しない変更がなされていないかのテスト」の項目は挙げていません。リグレッションテストで変更箇所が他に悪影響を及ぼしていないかの確認をしています。
54	変更要求仕様書とその上位(要求仕様、追加要求仕様・・・)とのトレーサビリティはとっていますか？(USDMの中で自然にとられているとの理解でしょうか)	当日会場でも回答しましたが、変更要求は変更要求仕様書にそのまま転記されるため、特にトレーサビリティは取っていません。
55	MBDとの関係についてどのようなことを考えているかご教示ください	XDDP、SPLいずれも開発プロセスの中でトレーサビリティを確保しながら設計を進めていく必要があります。設計の見える化をしつつトレーサビリティを確保していけるよう設計プロセスをモデル化していくべきだと考え、UMLをベースに独自の設計モデルを確立したいと考えています。
56	開発規模によるXDDPの導入の困難度の違いはあったでしょうか？規模が大きい場合には仕様整理、影響確認等がより大変になるのではというイメージがあります。	規模の大小でXDDPのプロセスが変わることはありません。ただし、大規模になるほど複数の要求仕様を複数の人間が分担するため、プロジェクト管理面の比重が大きくなります。

57	XDDPで開発を行っているときに 機能仕様書の記述は差分しか作成しないのですか？それとも、すべて反映したものになるのでしょうか？	XDDPでは変更内容単位に変更設計書(どう修正するか)を作成しますが、動作確認後に、原本の機能仕様書や設計仕様書のメンテナンスを行います。(原本があればですが・・)
58	仮説の実証で「担当者が経験者でなくても派生開発が可能とありますが実証に用いた指標について差し支えなかったら教えてください	指標としては特に定義しませんでした。開発体制で改造担当者の改造母体のノウハウを確認したレベルです。
59	テスト工程で工数が減少していますがその内訳について差し支えなければ紹介をお願いします。	テスト工程にはテストで抽出した不具合を修正する手戻り作業も含んでいます。XDDP適用によりこの手戻り作業が減っています。
60	進化型SPLはすでに適用例があります。Salion社(米国)の例で、まず単品開発してからそれを基にプロダクトライン(製品系列)を展開したものです。	「まず単品開発して」というのも進化型SPLとはちょっと違うように思いますが、参考になりそうですので調査してみます。
61	実績のある既存ソフトの流用、改善には、ブラックボックスのソフトから3点セットをおこすところから行われたのか。それは開発の部隊が行われたのでしょうか？	変更要求からブレークダウンする変更仕様と変更TM、変更設計書はブラックボックス状態のソフトを解析して改造を実施する設計部隊が作成しています。生産技術部(SEPG)としては、XDDP適用に際して各種支援をしていますが、実際の成果物は作成していません。

藤井講師

62	発電用DSL:プラントテーブル方式の逆ジェネレーションについて、試験で調整が必要となった場合、データベース(オブジェクト)を直接修正して逆ジェネレーションするということでしょうか？データベース(ドキュメント)を修正してジェネレーションするアプローチに比較したデメリットはないのでしょうか？ アナロジーとして、いきなりソースコードで不具合を修正して、そこからドキュメントをリバースで生成しているような悪いイメージを持ちました。	現地等で緊急な対応になった場合には、DBを直接修正するケースもあります。まず、本修正についてはDB(DSL)の修正に限定しており、コア資産については対象外です。コア資産に対しては、直接修正は実施しません。DSLについては影響範囲が限定されるように定義されており、通常のソフトの修正のように影響範囲が予想以上に広がることはありません。また、DSLを修正する場合にも、コア資産の一部として備えているオフラインのシミュレーション機能を用いて機能試験を実施した上で実際の使用に供しますので、プラント側にとんでもない影響が出てしまうことはありません。
----	---	--