

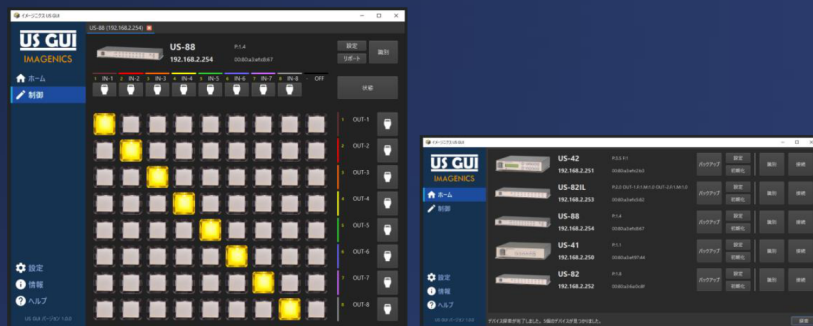
派生開発カンファレンス2025

23 May 2025, Yokohama, Japan

業務用機器ソフトウェア新規開発における 派生開発思考の活用による効率化戦略

～機種バリエーションを見据えた設計で、開発負荷低減と製品付加価値向上を両立～

ソフトウェア



成果物のユーザー企業数: 470社 (2025年4月24日現在)

ハードウェア



開発着手時の対象ハードウェア: 5機種



中村 晋一郎

結論

結論

派生開発手法は新規開発にも大変有用

- 派生開発の「機能追加や修正に必要な部分を理解し、変更仕様を定義して、影響範囲を明確化する手順と手法」は、リスクの大幅低減、開発負荷の低減と分散に役立つ。
- 新規開発対象に近い成果物を変更前の既存成果物として定義、新規開発を派生開発として扱う事で、効率的な開発が可能。

結論

何をやったら効果があったか？

- ▼組織的なこと

- ”やる”と決める (検討に時間を使わない)
- 最初から正しさを求めすぎない

- ▼派生開発的なこと (3種の神器+α)

- **XDDP**を知る / **PFD**でプロセスを定める / **USDM**で仕様を書く
- 新規開発対象に近い成果物を**変更前の既存成果物**として定義
- 変更要求に対して(既存実装の代わりに)**アーキテクチャ要素を割り当て**

新規開発対象に近い成果物を
変更前の既存成果物として定義

変更要求に対して(既存実装の代わり
に)アーキテクチャ要素を割り
当てTMで分析。



なぜ派生開発？

派生開発手法は新規開発にも大変有用

- 「新規開発だから完璧なものを最初から作らなければ」と過剰に構えると、かえって何も進まない。派生開発のように“今あるものをベースに差分で積み上げる”という発想を取り入れることで、適切なスタートと現実的な成果を獲得できるから。

開発背景

リベラルロジック株式会社

- **経営理念**

ソフトウェア開発を通じて、
顧客と良質な物語を創造し、
競争力のある日本を取り戻す。

- **使命**

顧客のニーズに応える
革新的で品質の高いソフトウェアを、
真の課題解決策と共に提供する。

- **存在意義**

中長期的視点に立った価値提供と
人材育成することで、
次世代の繁栄に貢献する。

良質な物語を
創造する関係
性を最重要視

ハードウェアの概要

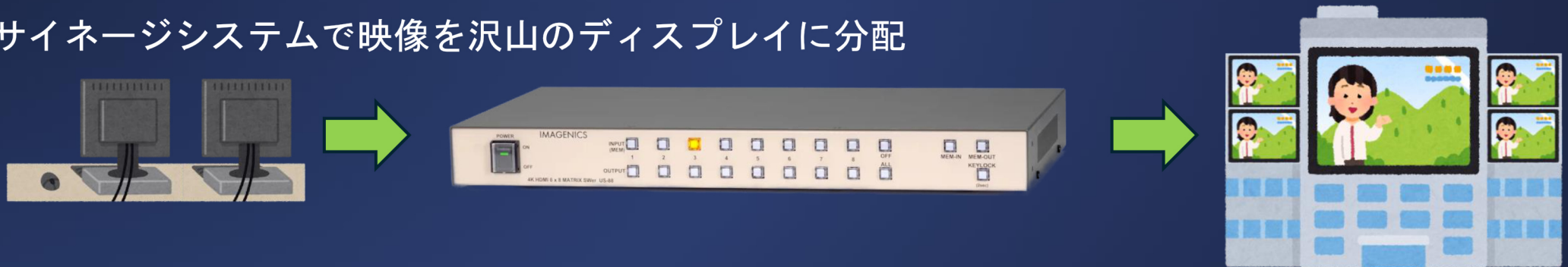
対象ハードウェアの概要

信頼性を要求される業務用途に特化した映像切替装置

監視システムで沢山のカメラ入力から切り替えて表示

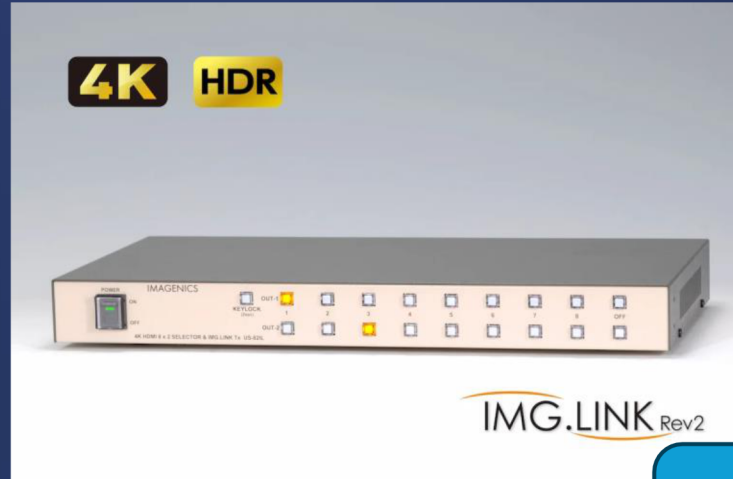
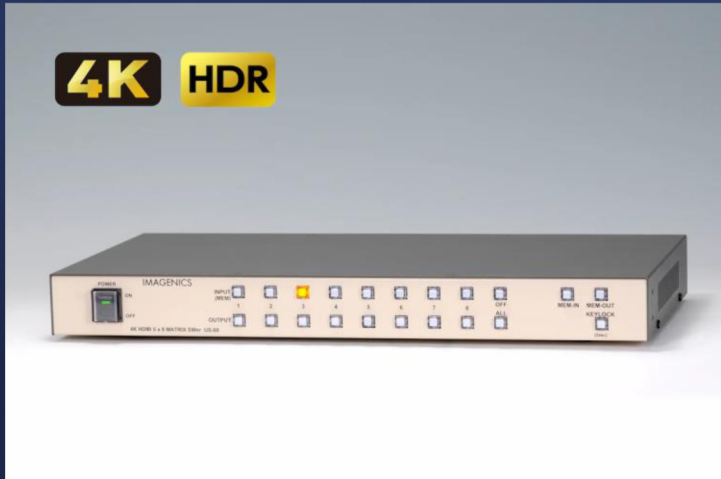


サイネージシステムで映像を沢山のディスプレイに分配



対象ハードウェアの概要

開発着手時の対象ハードウェア: 5機種



HDMI信号に対応した
映像信号スイッチャー

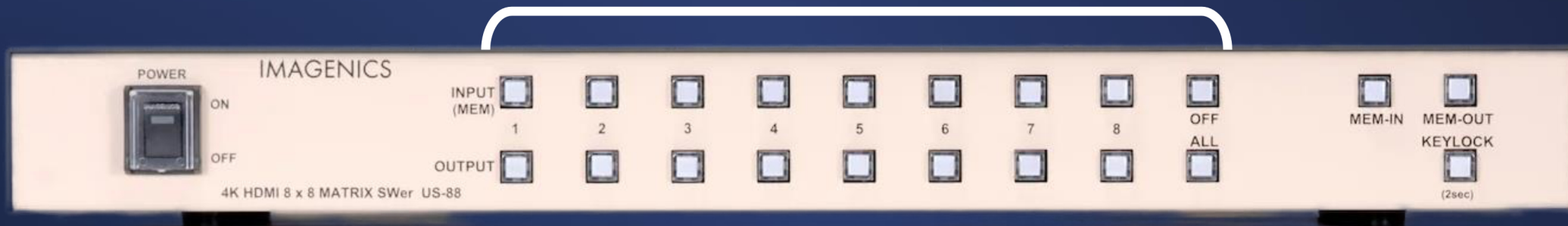
業務用システムに必要な
各種機能をコンパクトに
搭載した高品質な製品群

拘り抜いた品位の高い挙
動と抜群の安定性でクリ
ティカルな現場に最適

対象ハードウェアの概要

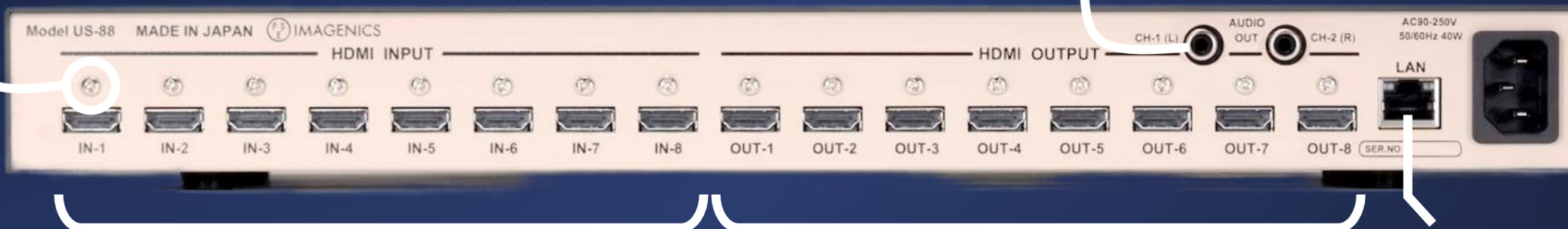
フロントパネルとリアパネルのインターフェース

入出力切替操作ボタン



HDMIケーブルをきっちり固定可能

アナログ音声出力



入力端子

出力端子

ネットワーク
通信端子

切替基本操作は本体正面パネルで操作
細かな設定はネットワーク経由で

ハードウェアから
明示的/暗黙的に
得られた要請

細かな設定はネットワーク経由で

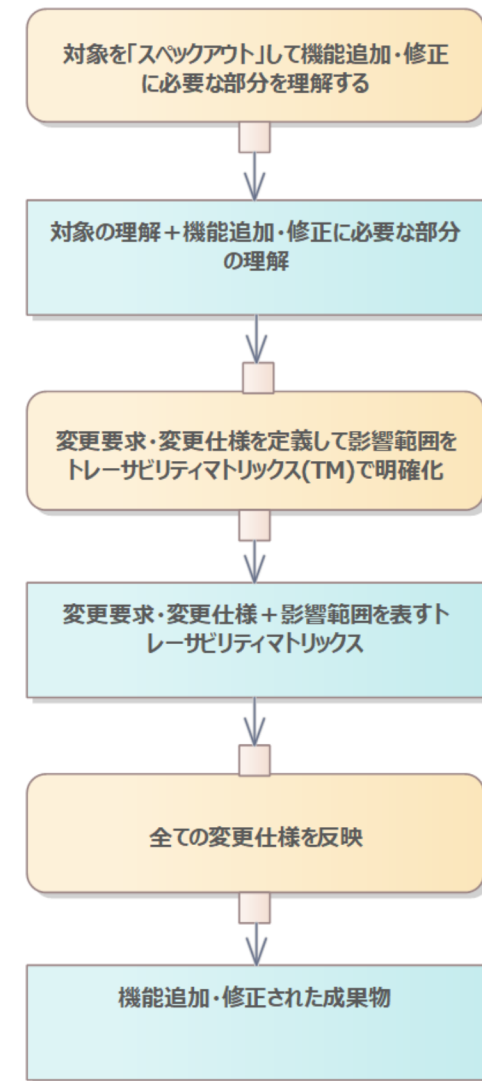
派生開発手法の 新規開発への応用可能性

新規開発への応用可能性

既存成果物と見做すものを用意

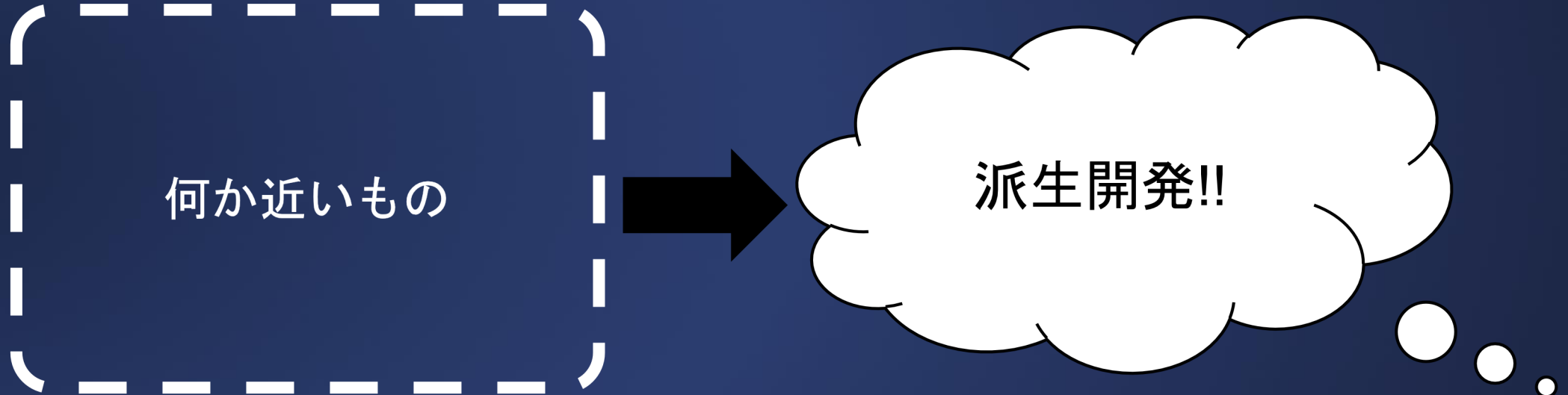
- 派生開発の「機能追加・修正に必要な部分を理解し、変更仕様を定義して、影響範囲を明確化する手順と手法」は、リスクの大幅低減、開発負荷の低減と分散に役立つ。
- 新規開発対象に近い成果物を、変更前の既存成果物として定義。
- 変更要求に対して(既存実装の代わりに)アーキテクチャ要素を割り当てトレーサビリティマトリックス(TM)で分析。

AFFORDD派生開発の基本的な特徴



新規開発への応用可能性

新規開発対象に近い既存成果物を
《変更前の成果物》として定義



向かうべき方向性が明確になり、新規開発にありがちなアイデアや方向性の発散を防止。

良い意味で開発に対する拘束条件の追加

既存成果物

ベースとなるシステム

既存成果物定義のポイント

ベースとなるシステムを定義するには？

◆ 見た目や機能よりも、論理的に近い位置にあるかが重要

- 近い例: 通信機能はあるが、抽象化されていない。
- 近い例: 現状コマンドラインツールだが、内部はコアとUIに分離可能。
- 一見距離のありそうな成果物でも、本質的に近い場合もある。

※要求とアーキテクチャに大きな影響を与える本質的な論理要素は、見た目が関連する前の段階で既に性格づけられていることが多い。

◆ 抽出した論理構成で最終成果物に向かう論理矛盾が無いこと

- 遠い例: 「ロケット本体」を作ろうとしている時に、
「ロケット発射台」をベースシステムにするのは無理。
- 近い例: 「簡単操作作用ソフト」を作るのに、
「通信層を備えた既存ソフト」はベースシステムになるかも。

既存成果物の定義

新規開発対象に近い成果物を変更前の既存成果物として定義

- 取扱説明書で案内されていた「コマンドを打つ環境」を「変更前の既存成果物」として定義。
 - ユーザーが実際に使用していた環境であるから適切と判断。
- 内部実装は提供されないなので、外部作用を参考に、機能と内部構造を整理。(推察含む)
 - 派生開発の「変更前の既存成果物」の役目を果たせば良いと割り切り。

既存成果物の難点（開発に至る動機）

“打てるコマンドがたくさんある / 機種によって無いものもある”

機能	US-88	US-82IL	US-82	US-42	US-41
探索	ver	ver/IA	ver/IA	ver	ver
バックアップデータ保存	bA	bA	bA	bA	bA
バックアップデータ復旧	Ba	Ba	Ba	Ba	Ba
IPアドレス設定	Nc/Nd/Ne/Nf	Nc/Nd/Ne/Nf	Nc/Nd/Ne/Nf	Nc/Nd/Ne/Nf	Nc/Nd/Ne/Nf
IPアドレス取得	nC/nD/nE/nF	nC/nD/nE/nF	nC/nD/nE/nF	nC/nD/nE/nF	nC/nD/nE/nF
サブネットマスク設定	Ng/Nh/Ni/Nj	Ng/Nh/Ni/Nj	Ng/Nh/Ni/Nj	Ng/Nh/Ni/Nj	Ng/Nh/Ni/Nj
サブネットマスク取得	nG/nH/nI/nJ	nG/nH/nI/nJ	nG/nH/nI/nJ	nG/nH/nI/nJ	nG/nH/nI/nJ
ゲートウェイ設定	Nk/NI/Nm/Nn	Nk/NI/Nm/Nn	Nk/NI/Nm/Nn	Nk/NI/Nm/Nn	Nk/NI/Nm/Nn
ゲートウェイ取得	nK/nL/nM/nN	nK/nL/nM/nN	nK/nL/nM/nN	nK/nL/nM/nN	nK/nL/nM/nN
工場出荷設定初期化	Ma/Mb	Ma/Mb	Ma/Mb	Ma	Ma/Mb
識別(キーロックインジケータ流用)	Hb	Hb	Hb	Hb	Hb
接続(モデル名取得)	ver	ver/IA	ver/IA	ver	ver
スイッチャ状態制御	Xa/Xb/Xc/Xd/Xe/Xf/Xg/Xh	Xa/Xb/Xc/Xd/Xe/Xf/Xg/Xh	Xa/Xb/Xc/Xd/Xe/Xf/Xg/Xh	[Input].[Output]	Xa/Xb/Xc/Xd
スイッチャ状態取得	xA/xB/xC/xD/xE/xF/xG/xH	xA/xB/xC/xD/xE/xF/xG/xH	xA/xB/xC/xD/xE/xF/xG/xH	z[Output]	xA/xB/xC/xD
クロスポイントメモリ読み込み	Ta	(本体機能なし)	(本体機能なし)	(本体機能なし)	(本体機能なし)
クロスポイントメモリ書き込み	Tb	(本体機能なし)	(本体機能なし)	(本体機能なし)	(本体機能なし)
リポート作成	rA	rA	rA	rA	rA
入出力信号状態取得	rA	rA	rA	rA	rA
キーロック設定・取得	Ha/hA	Ha/hA	Ha/hA	Hb/hB	Ha/hA
キーロックモード設定・取得	(本体機能なし)	(本体機能なし)	(本体機能なし)	Ha/hA	(本体機能なし)
起動時クロスポイントモード設定・取得	Hb/hB	Hb/hB	Hb/hB	Hc/hC	Hb/hB
ビデオオフモード	(本体機能なし)	(本体機能なし)	(本体機能なし)	Hd	(本体機能なし)
シーケンシャルスイッチ時間間隔	(本体機能なし)	(本体機能なし)	(本体機能なし)	He	(本体機能なし)
ビデオバイパスモード設定・取得	Hc/hC	Hc/hC	Hc/hC	(本体機能なし)	Hc/hC
オーディオの分離信号モード設定	(本体機能なし)	(本体機能なし)	(本体機能なし)	Ad/Ae/Af/Ag	(本体機能なし)
オーディオの分離信号モード取得	(本体機能なし)	(本体機能なし)	(本体機能なし)	aD/aE/aF/aG	(本体機能なし)

機能	US-88	US-82IL	US-82	US-42	US-41
オーディオの分離信号レベル設定	(本体機能なし)	(本体機能なし)	(本体機能なし)	Ah/Ai/Aj/Ak	(本体機能なし)
オーディオの分離信号レベル取得	(本体機能なし)	(本体機能なし)	(本体機能なし)	aH/al/aJ/aK	(本体機能なし)
オーディオの分離信号遅延時間設定	(本体機能なし)	(本体機能なし)	(本体機能なし)	Ac	(本体機能なし)
オーディオの分離信号遅延時間取得	(本体機能なし)	(本体機能なし)	(本体機能なし)	aC	(本体機能なし)
HDMI入力ポートEDID設定	eA/eB/eC/eD/eE/eF/eG/eH	eA/eB/eC/eD/eE/eF/eG/eH	eA/eB/eC/eD/eE/eF/eG/eH	iA/iB/iC/iD	eA/eB/eC/eD
HDMI入力ポートEDID取得	Ea/Eb/Ec/Ed/Ee/Ef/Eg/Eh	Ea/Eb/Ec/Ed/Ee/Ef/Eg/Eh	Ea/Eb/Ec/Ed/Ee/Ef/Eg/Eh	la/lb/lc/lD	Ea/Eb/Ec/Ed
HDMI入力ポートHDCP設定	dA/dB/dC/dD/dE/dF/dG/dH	dA/dB/dC/dD/dE/dF/dG/dH	dA/dB/dC/dD/dE/dF/dG/dH	iE/iF/iG/iH	dA/dB/dC/dD
HDMI入力ポートHDCP取得	Da/Db/Dc/Dd/De/Df/Dg/Dh	Da/Db/Dc/Dd/De/Df/Dg/Dh	Da/Db/Dc/Dd/De/Df/Dg/Dh	le/lf/lg/lh	Da/Db/Dc/Dd
HDMI入力ポートカラーフォーマット取得	(本体機能なし)	(本体機能なし)	(本体機能なし)	li/lj/lk/lI	(本体機能なし)
HDMI入力ポートカラーフォーマット設定	(本体機能なし)	(本体機能なし)	(本体機能なし)	il/iJ/iK/iL	(本体機能なし)
アナログオーディオ入力重畳信号モード	(本体機能なし)	(本体機能なし)	(本体機能なし)	Aa/aA	(本体機能なし)
アナログオーディオ入力重畳信号レベル	(本体機能なし)	(本体機能なし)	(本体機能なし)	Ab/aB	(本体機能なし)
シーケンシャルスイッチモード設定	(本体機能なし)	Sa/Sb	Sa/Sb	Og/Oh	Sa/Sb
シーケンシャルスイッチモード取得	(本体機能なし)	sA/sB	sA/sB	oG/oH	sA/sB
シーケンシャルスイッチ時間間隔設定	(本体機能なし)	Qa/Qb	Qa/Qb	He	Qa/Qb
シーケンシャルスイッチ時間間隔取得	(本体機能なし)	qA/qB	qA/qB	hE	qA/qB
出力フォーマット設定	(本体機能なし)	(本体機能なし)	(本体機能なし)	Oa/Ob	(本体機能なし)
出力フォーマット取得	(本体機能なし)	(本体機能なし)	(本体機能なし)	oA/oB	(本体機能なし)
出力カラーモード設定	(本体機能なし)	(本体機能なし)	(本体機能なし)	Oc/Od	(本体機能なし)
出力カラーモード取得	(本体機能なし)	(本体機能なし)	(本体機能なし)	oC/oD	(本体機能なし)
ディープカラーモード設定	(本体機能なし)	(本体機能なし)	(本体機能なし)	Oe/Of	(本体機能なし)
ディープカラーモード取得	(本体機能なし)	(本体機能なし)	(本体機能なし)	oE/oF	(本体機能なし)

既存成果物の難点（開発に至る動機）

“同じコマンドでも違う意味を採用している機種がある”

起動時のスイッチの状態を設定したい

Hbに続いてパラメータ
➡ Hcに続いてパラメータ（一部機種）

ビデオバイパスモードを設定したい

➡ Hcに続いてパラメータ
機能なし（一部機種）

コマンドは機械のために作られている
人間が打つようには作られていない

既存成果物の難点（開発に至る動機）

“設定したIPアドレスは...メモしておかないと...”の難しさ

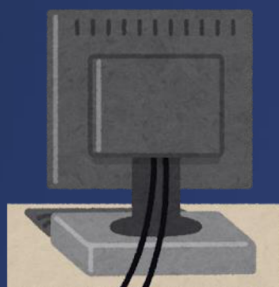
IPアドレスのメモを無くしちゃった！
上から順に192.168.11.10, 192.168.11.11,
192.168.11.12, ...えーっと本当にそうだったけ？

~~192.168.11.20~~

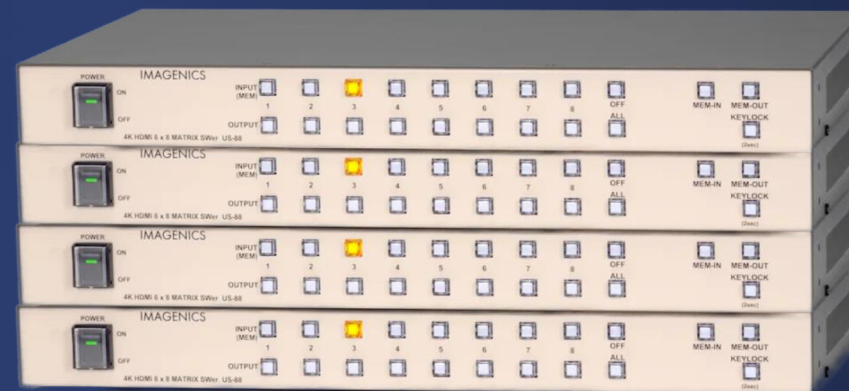
本体に表示デバイス
が無い機種が存在



利用者



操作端末



操作したい機器

ネットワーク設定を忘れてしまうと設定できない。
現場に複数台の機器がある場合、なおさら混乱。

既存成果物の難点（開発に至る動機）

凄く良いハードウェアなのに難所が辛い...

ユーザー



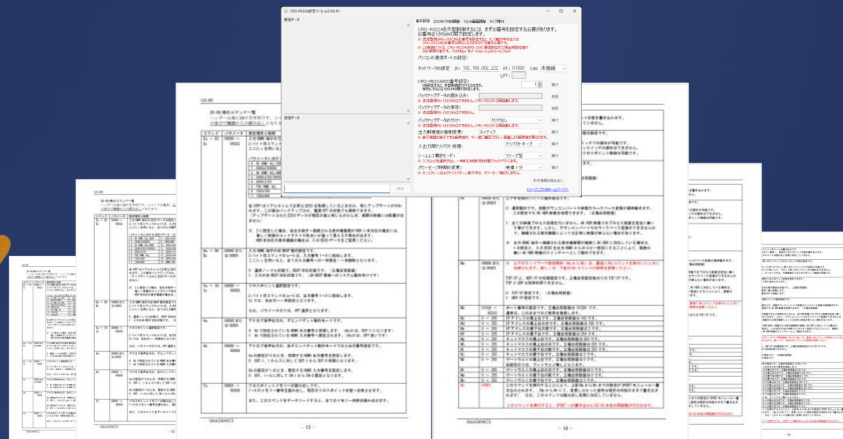
巨大な山脈



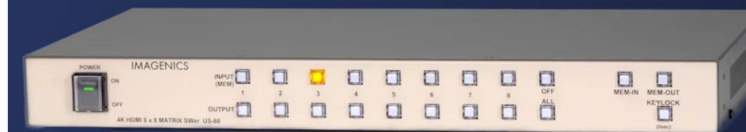
目的地



使う可能性がゼロではない人達



コマンド表とコマンドツール



操作したい機器

既存成果物の難点（開発に至る動機）

コンセプトの策定

ハードウェアから
明示的/暗黙的に
得られた要請

切替基本操作は本体正面パネルで操作
細かな設定はネットワーク経由で



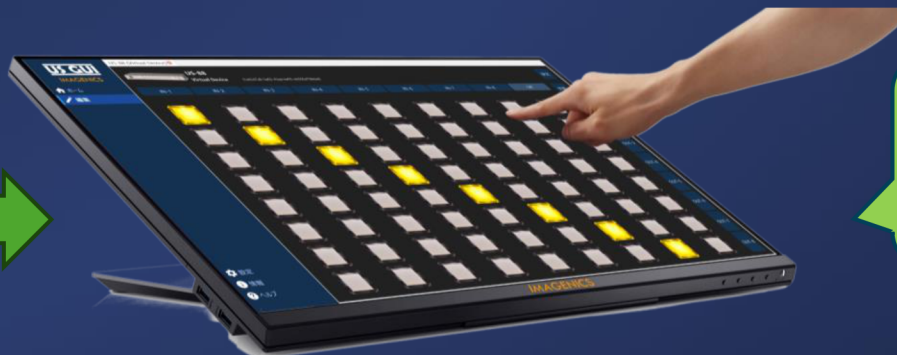
種々状況をふまえ開発する価値を予め定義し、誰にも理解できる平易な言葉で言語化。
全利害関係者の目的地を明確にし、開発の主眼がブレないようにイメージを固定。

ソフトウェア開発
前に策定した開発
コンセプト

見つける・つながる・動かせる

メモっていなくても見つけて、見つけるから簡単に繋がられて、
繋がれるから動かせて、誰も困らない便利なソフトウェア！

開発開始時に制作した
コンセプトアート画像



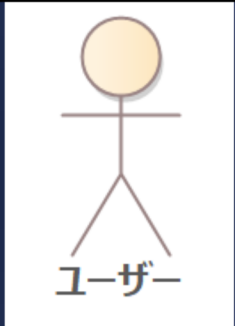
開発の主眼が
ブレないように
イメージを固定

要求定義プロセス

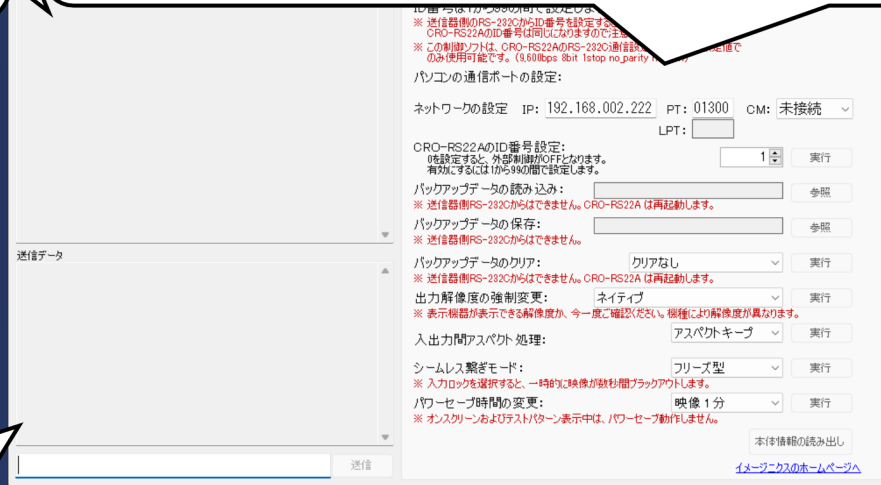
要求定義プロセス

現状の理解 (既存成果物でユーザーが行なう事を列挙する)

制御ソフトウェアは、制御対象から受信したデータを受信コマンド列挙表示部に表示する。



ユーザーは、接続対象入力部に接続対象IPアドレスとポート番号を入力する。

A screenshot of a web-based control interface. It features a '送信データ' (Transmit Data) section on the left and a configuration section on the right. The configuration section includes fields for 'ネットワークの設定' (Network Settings) with IP, Port, and CM, and 'CRO-RS22AのID番号設定' (CRO-RS22A ID Number Setting). There are also sections for 'バックアップデータの読み込み' (Load Backup Data), 'バックアップデータの保存' (Save Backup Data), 'バックアップデータのクリア' (Clear Backup Data), '出力解像度の強制変更' (Force Output Resolution Change), '入出力間アスペクト処理' (Aspect Ratio Processing), 'シームレス繋ぎモード' (Seamless Connection Mode), and 'パワーセーブ時間の変更' (Change Power Save Time). Each section has a '実行' (Execute) button.

制御ソフトウェアは、ユーザーが入力したコマンドを制御対象に送信する。



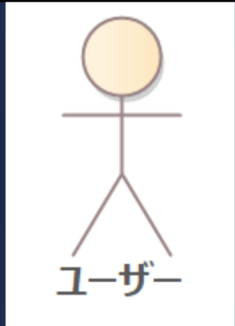
制御ソフトウェアは、ユーザーが入力したコマンドを送信コマンド列挙表示部に表示する。

ユーザーは、取扱説明書に書かれたコマンドをコマンド入力部に入力し、送信ボタンを押す。

要求定義プロセス

現状の理解 (既存成果物の違和感のある体験を抽出する)

制御ソフトウェアは、制御対象から受信したデータを受信コマンド列挙表示部に表示する。



ユーザーは、接続対象入力部に接続対象IPアドレスとポート番号を入力する。

何が有れば
ユーザーが
ラクになるか

制御ソフトウェアは、ユーザーが入力したコマンドを制御対象に送信する。



制御ソフトウェアは、ユーザーが入力したコマンドを送信コマンド列挙表示部に表示する。

ユーザーは、取扱説明書に書かれたコマンドをコマンド入力部に入力し、送信ボタンを押す。

[操作に必要なモノコト]

- 機種名称や型式
- ユーザーマニュアル
- コマンドに対する知識
- IPアドレス / ポート番号
- ツール操作方法

要求定義プロセス

現状の理解 (既存成果物の違和感のある体験を抽出する)

- 本体フロントパネルにIPアドレスを表示できない機種もある
- コマンド(英数字の羅列)をユーザーが間違えずに打つのは困難
- ユーザーマニュアルが無ければコマンドを打てない
- 直感的でないので操作に時間がかかる/やる気がでない

▼欲しいモノ

IPアドレスを知らずとも機器と接続が可能で、
コマンドを知らずとも操作を実行でき、
ユーザーマニュアルを見なくても、
直感的に操作できるユーザーインターフェース、

仕様策定プロセス

仕様策定プロセス

変更要求と変更仕様の関係

変更要求（Change Request）

- ユーザー（顧客や市場）からの「こうして欲しい」「こうだったら便利」という要望や期待。
- 業務課題・不具合の指摘・機能追加など、上流のニーズベース。
- 顧客、営業、運用チームなど、製品を利用する／届ける立場から出てくる。

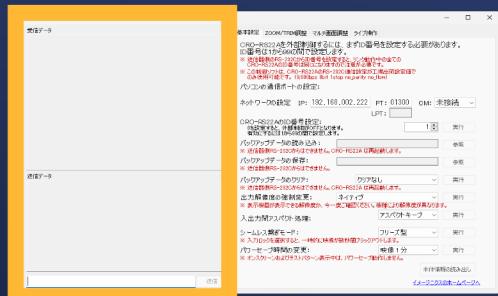
変更仕様（Change Specification）

- 変更要求に対して「このように変更する」という、設計者視点の具体的な変更内容。
- アーキテクチャやモジュール構成の中で「どこをどう変えるのか」を設計対象の中で定義。
- 影響範囲を特定し、適切に局所的な変更を施すことで、品質とコストをバランスする。

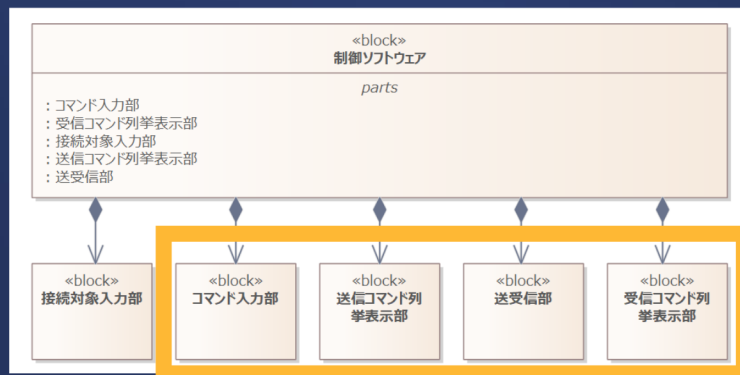
仕様策定プロセス

既存成果物を分析把握

内部実装は提供されないので、外部作用を参考に、機能と内部構造を整理。(推察含む)
派生開発の「変更前の既存成果物」の役目を果たせば良いと割り切り。

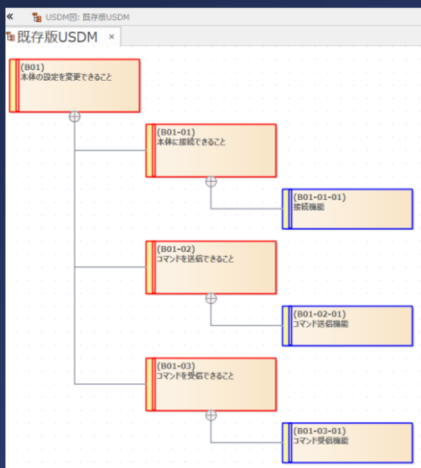


「コマンドを打つ」
を実現する機能群



「コマンドを打つ」
を実現する機能群

既存システムは、ユーザーにコマンドを打ってもらう事で、システムの状態を変えて、ユーザーの要求を満たす状況を作り出すもの。



ソース	1. 《現状分析》対象をバックアウツして機能追加・修正に必要な部分を理解する	既存版	要求	仕様	制御ソフトウェア (既存版)	コマンド入力部	受信コマンド列挙表示部	接続対象入力部	送受信部	送信コマンド列挙表示部
1. 《現状分析》対象をバックアウツして機能追加・修正に必要な部分を理解する	13	13	0	0	1	2	2	2	4	2
既存版	13	13	0	0	1	2	2	2	4	2
要求	7	7	0	0	1	1	1	1	2	1
本体の設定を変更できること	1	1	0	0	↑					
コマンドを受信できること	2	2	0	0		↑			↑	
コマンドを送信できること	3	3	0	0		↑			↑	↑
本体に接続できること	1	1	0	0			↑			↑
仕様	6	6	0	0	0	1	1	1	2	1
コマンド受信機能	2	2	0	0			↑			↑
コマンド送信機能	3	3	0	0		↑			↑	↑
接続機能	1	1	0	0				↑		

既存成果物の主体は
「コマンドを打つ」にある。

仕様策定プロセス

既存成果物の改善策考案

アーキテクチャの下層が提供するサービスによって、意図せず製品像が形作られてしまう事が往々にしてある。この既存成果物の場合、通信する機能が上位に露呈していたために、ユーザーが実装詳細を知らなければならなかった。ユーザー界面に何を配置するのが非常に重要。

▼欲しいモノ

IPアドレスを知らずとも機器と接続が可能で、コマンドを知らずとも操作を実行でき、ユーザーマニュアルを見なくても、直感的に操作できるユーザーインターフェース、

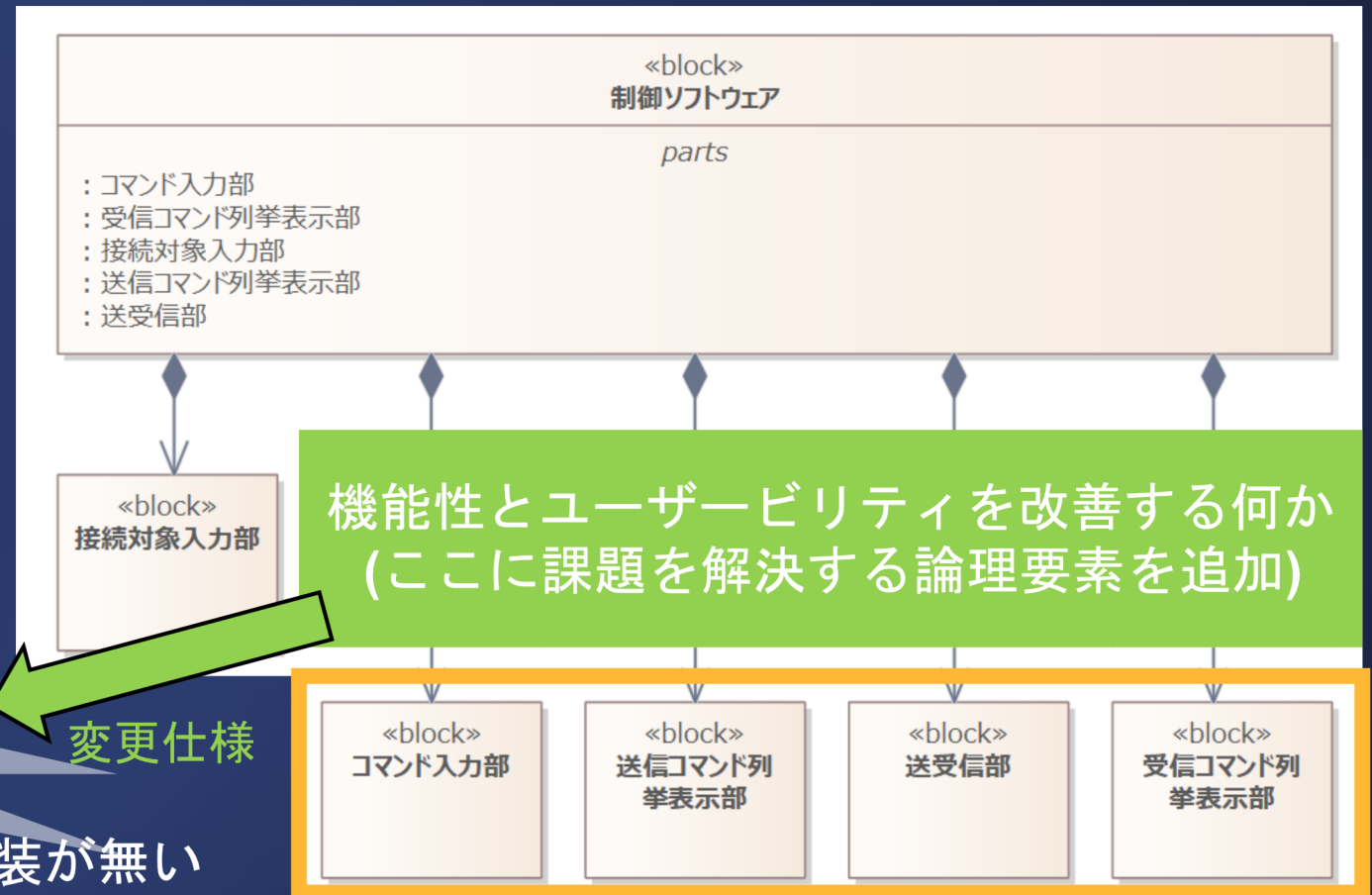
変更要求

変更要求＝欲しいモノ

変更仕様＝アーキテクチャの論理部品

変更仕様

👉 新規設計の場合、分析対象となる実装が無いので、代わりにアーキテクチャを割り当てる



ユーザーにとっての煩雑さ要因

変更要求と変更仕様

変更要求の分析・設計を経て、変更仕様として具体化される流れ

1. ユーザーの声（欲しいモノ）
2. 変更要求（外から見た変更）

3. アーキテクト・開発者による分析と設計
4. システム内での変更仕様（中から見た変更）

USDM

2	要求	v0.1.0	SDK Version 0.1.0 デバイス探索から基礎制御機能まで一通り実現した初めてのバージョン
3	理由		先のバージョンは特定のアドレスを指定する事で接続制御するだけのものだった。デバイス探索の実現可能性が明確になったことから、デバイス探索から接続、接続から基礎制御実現までを実現し、その有効性を検証する。
4	説明		
5	要求	v0.1.0-01	インターフェースの抽象化
6	理由		先のバージョンまではインターフェースの抽象化がなされておらず、直接的な関係で構成されていたため他機種に渡る派生開発が難しかった。このインターフェースの抽象化によって具体実装は上位から隠蔽され、機種の派生が適切な設計で行なう事が可能になる。
7	説明		
8		☐ ☐ ☐	#2437 スイッチャの抽象化インターフェースの設計検証 【理由】 制御対象機器の各種機能群に対して、何らかのくくりで抽象化したインターフェースとして定義し、そのことで下層の実装は抽象化インターフェースを実装するだけで良い、という開発体制を実現する。これにより派生開発における検討完了後の一括実装を可能にできると判断したため。 【説明】 抽象化インターフェースの実装は、おそらくEquipmentが抽象化インターフェースを多量継承する形を取るだろう ユニットテストをtestswitcher, crosspointmemory.cppを参考に全てのEquipmentで動作させられるようにする
9		☐ ☐ ☐	#2452 デストの網羅性を全機種を対象にしたものにしていくため。 【理由】 現状の実装は固有機種を対象にしている。簡単な実装変更で全機種への対応が可能になると判断した。 【説明】 サンプルプログラムをSocketEquipmentとVirtualEquipmentの両方で動作させられるようにする 現状のVirtualEquipmentは下層に隠蔽されており、上位から意図して生成する事ができない。これでは仮想的な制御対象として使えないから。 【説明】 仮に上位から自由に仮想デバイスを作れるならば、本体がなくても上位アプリケーションの建設ができるようになる。
10		☐ ☐ ☐	#2467 DeviceDiscoveryの実装詳細を追加する

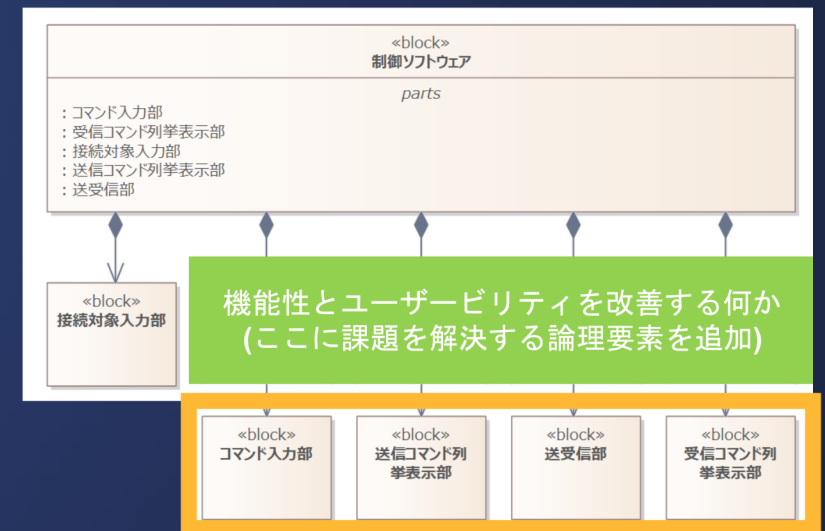
要求を満たす
アーキテクチャを考案

アーキテクチャが
要求を満たすか検証



トレーサビリティマトリックス(TM)

アーキテクチャ



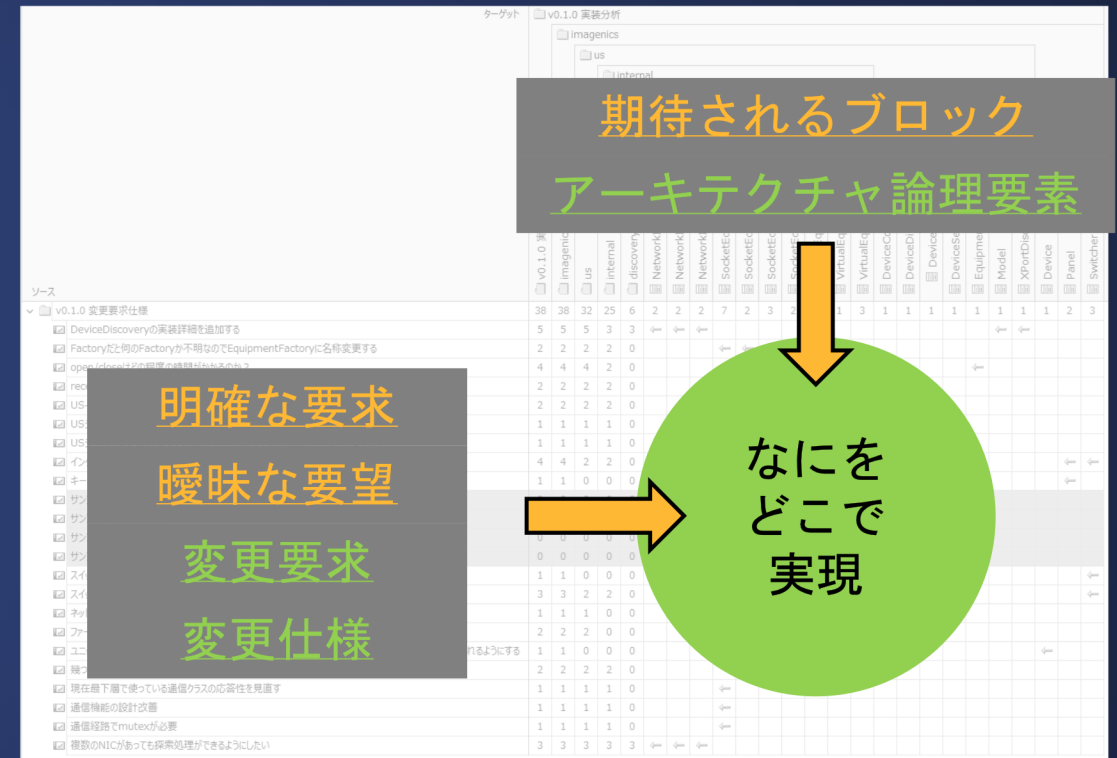
ユーザーにとっての煩雑さ要因

変更要求と変更仕様

トレーサビリティマトリックス(TM)活用の有用性

- 新しい要求を、アーキテクチャの何処で満たすようにするのか、短時間で検討できる。
- 複数の論理に跨る内容を統合したり分離したりする際に、精度の高い判断が可能。
- 明確な要求だけでなく、曖昧な要望を列挙しておくことで、見えない関係性に気付く。

※要求の矛盾に気付いたり、要求品質改善に繋がるアイデアの着想が、アーキテクチャを同時に見る事で得られる事もある。

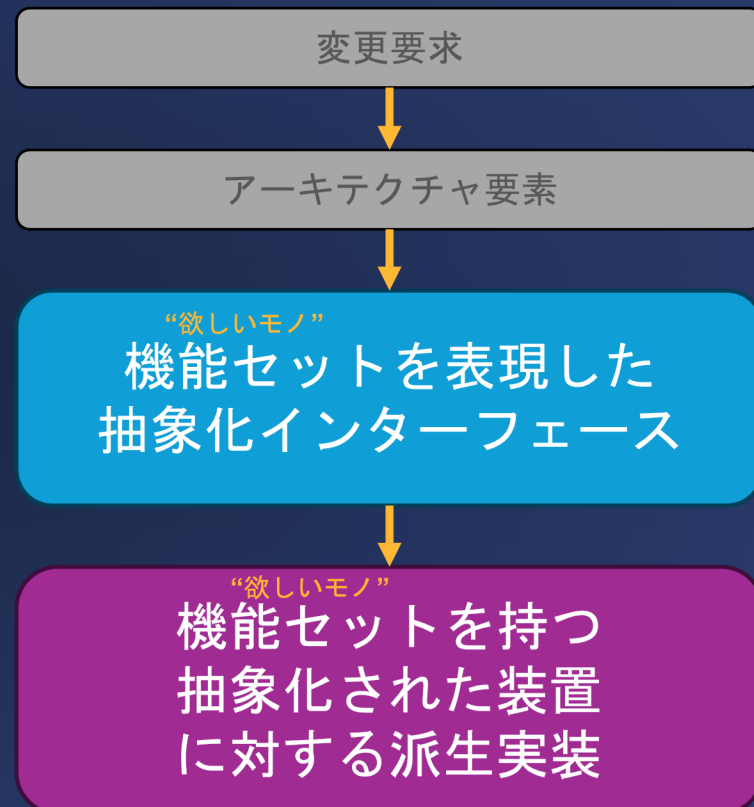


作ってからでは時間がかかって遅いから、
TMを活用して作る前に合理的な判断をする。

一括実装を前提とした 設計最適化とその効果

一括実装を前提とした設計

欲しいモノの実現に向かう論理階層構造

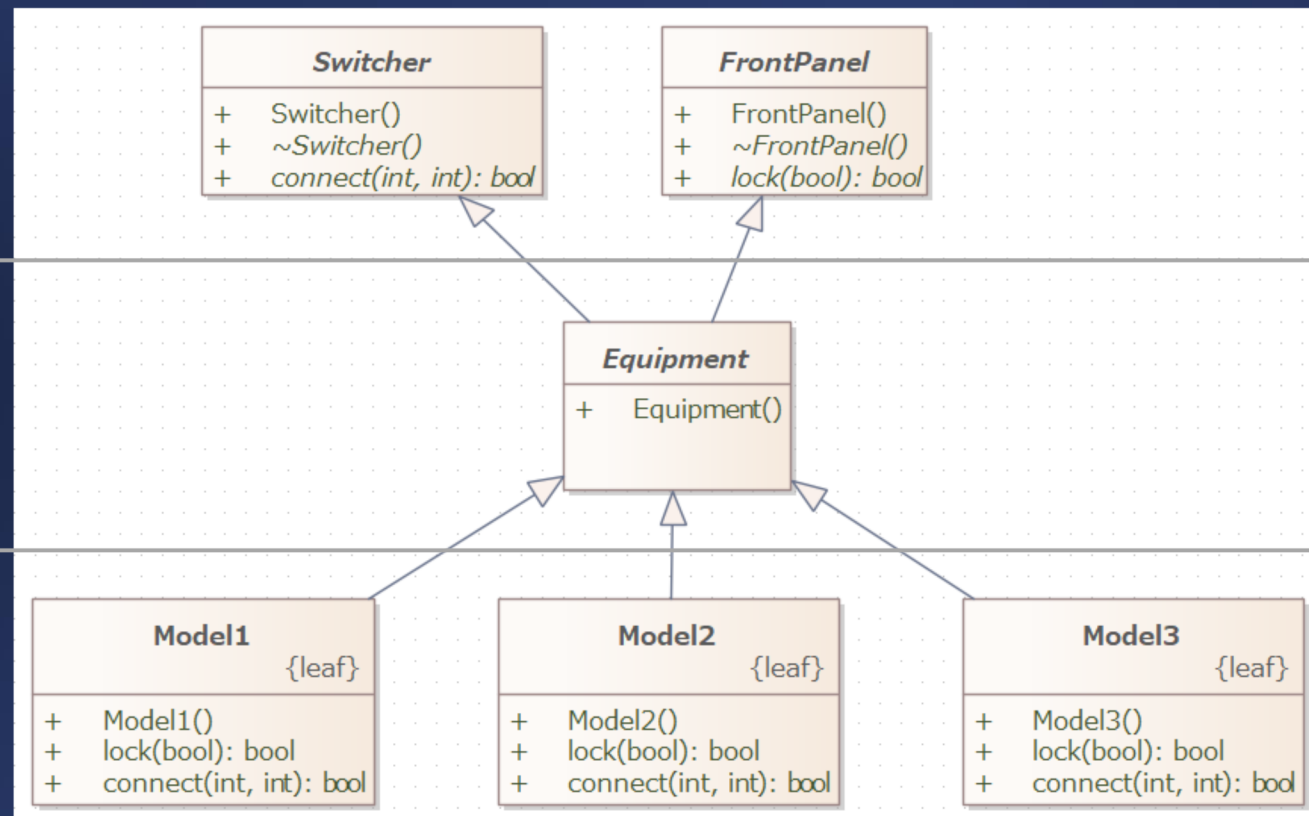


変更要求に対応したアーキテクチャ要素から導き出されるもの。“欲しいモノ”

アーキテクチャ要素に対する実装の提供。つまり”欲しいモノ”の実現。

一括実装を前提とした設計

欲しいモノの実現に向かう論理階層構造



“欲しいモノ”
機能セットを表現した
抽象化インターフェース

“欲しいモノ”
機能セットを持つ
抽象化された装置

“欲しいモノ”
機能セットを持つ
抽象化された装置
に対する派生実装

機能単位の差分開発が可能で、高い抽象度から具体実装まで、段階的詳細化を実現できる。
派生開発手法との相性が良く、清水氏が提唱しておられた「一括実装」にも大変に有用。

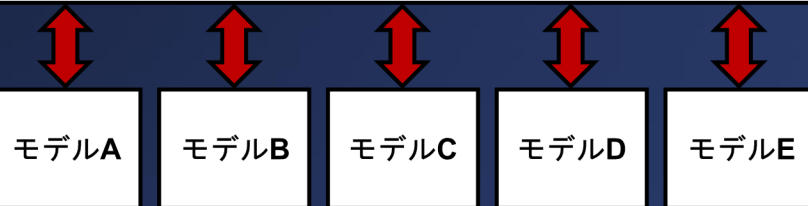
一括実装を前提とした設計

欲しいモノの実現に向かう論理階層構造

“欲しいモノ”

機能セットを持つ抽象化された「装置」
として論理を構成。上位からは等価に扱
いながらの多機種展開が非常に容易。

抽象化された「装置」
(上位からはこれしか見えない)



トレーサビリティマトリックス(TM)で整理した
* 変更要求=欲しいモノ
* 変更仕様=アーキテクチャの論理要素
を直交性の高い構造 (複雑度低)で実現可能。

“欲しいモノ”

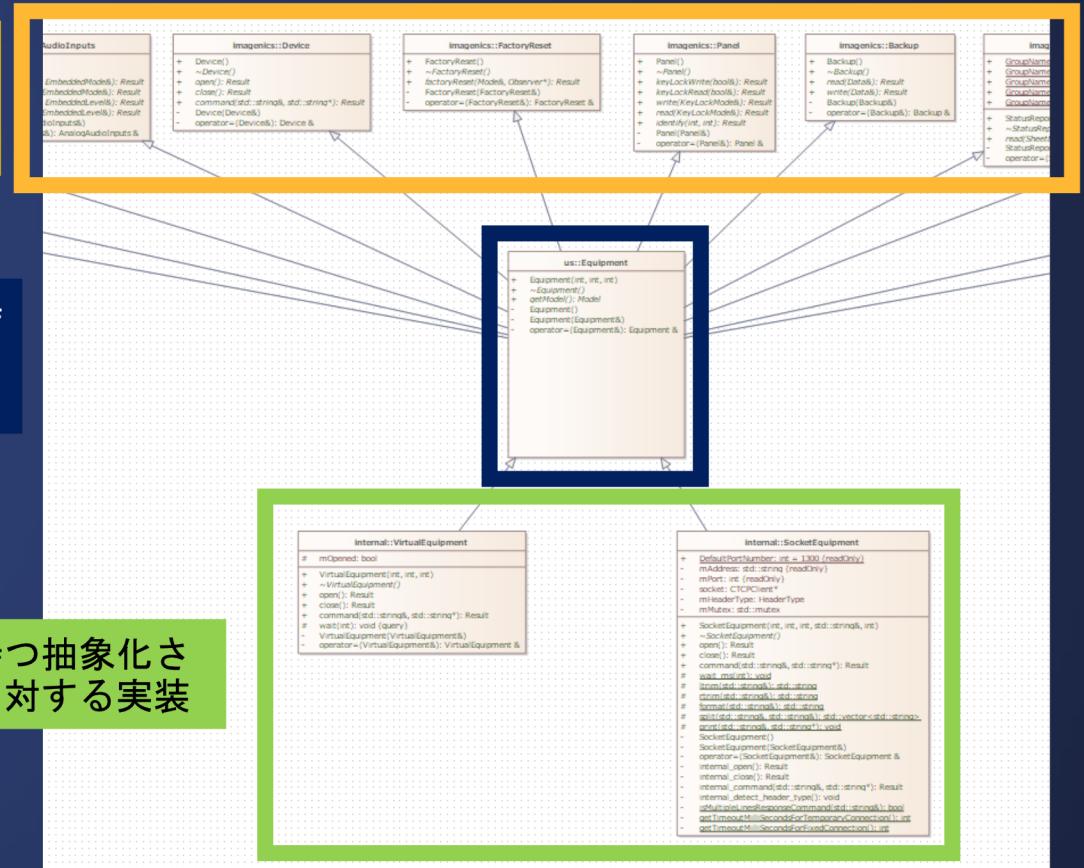
機能セット毎
のインター
フェース定義

“欲しいモノ”

機能セットを持
つ抽象化された
「装置」

“欲しいモノ”

機能セットを持つ抽象化さ
れた「装置」に対する実装



一括実装を前提とした設計

欲しいモノの実現に向かう論理階層構造

要望要求
アーキテクチャに近い



ソフトウェアの構造を、要望要求・
アーキテクチャに近いところから、
具体的な装置・実装に近いところへ
と、段階的に詳細化できる。
(二要素間をTMで確認できる)



具体的な装置
実装に近い

機能セット毎のインターフェース定義

“欲しいモノ” (概念的)

機能セットを持つ抽象化された「装置」

機能セットを持つ抽象化された
「装置」に対する実装 (第一層目)

機能セットを持つ抽象化された
「装置」に対する実装 (第二層目)

機能セットを持つ抽象化された
「装置」に対する実装 (第三層目)

“実現手段” (具体的)

「装置」が特定の抽象インターフェース定義を
継承した瞬間が、派生機種に対する実装要求。
抽象化インターフェース定義を継承した瞬間か
ら派生機種は機能を実装できる。
(清水氏の云う一括実装を言語機能で保証できる)

実装成果物と評価

実装成果物と評価

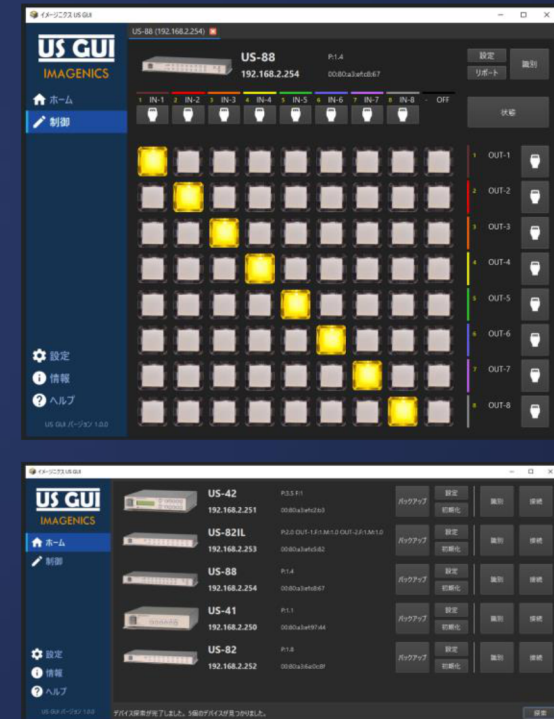
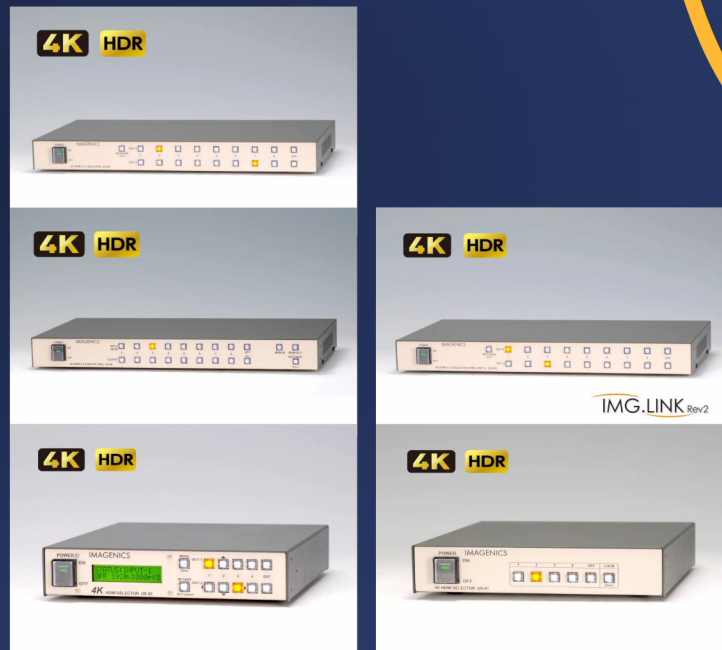
階層構造

ドメイン層
(ビジネス・ロジック)
SDK

アプリケーション・サービス層
(ユーザー・インターフェース)
GUI / CLI

User Interface

Core

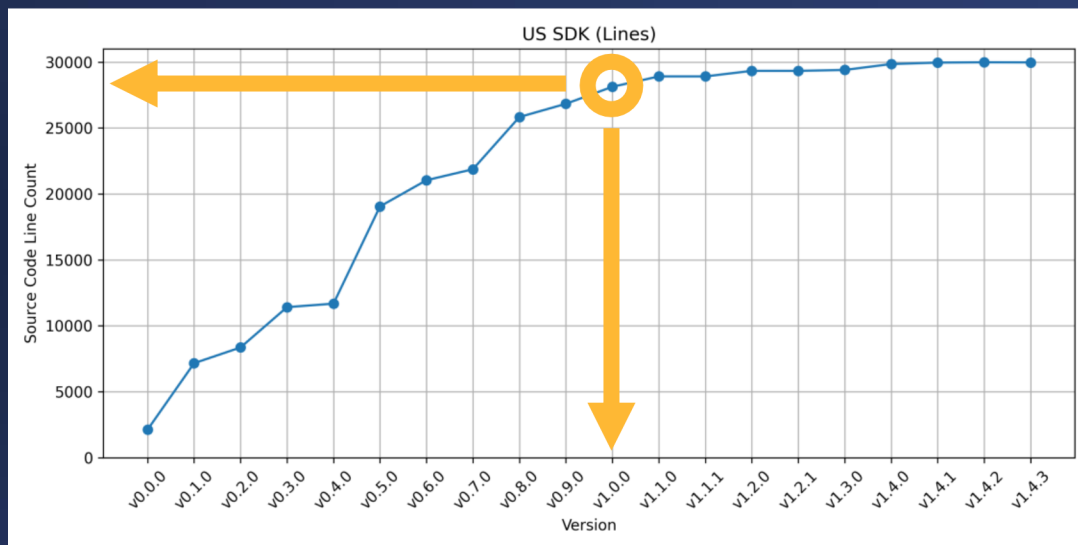


成果物はドメイン層とアプリケーション・サービス層の二層に分離して設計。

実装成果物と評価

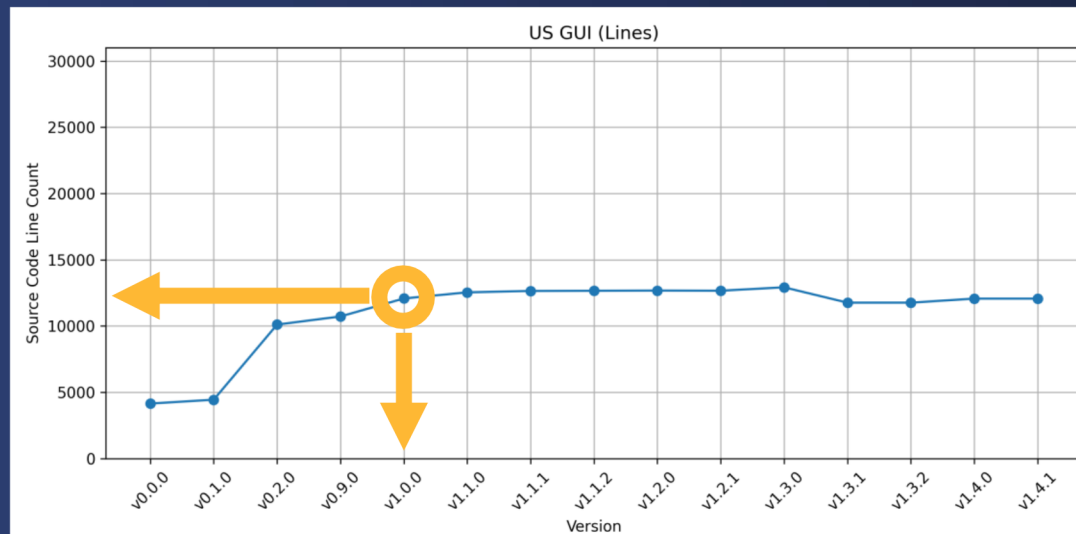
階層構造別開発規模

実装行数
ドメイン層



約28,000行

実装行数
アプリケーション・サービス層



約12,000行

合計約40,000行

(コメント行数: 約5%)

実装成果物と評価

開発規模

- 対象機器

US-88 / US-82IL / US-82 / US-42 / US-41 (実機)

US-88 / US-82IL / US-82 / US-42 / US-41 (仮想)

- 対象プラットフォーム

Windows / macOS

- 実装言語

C++

- 開発規模

約40,000行

- 開発工期

2022年8月09日 ~ 2022年10月11日 (71日間)

Version 0.0.0 ~ Version 1.0.0

実装成果物と評価

開発規模 [参考資料]

- 独立行政法人 情報処理推進機構 (IPA)の発表資料において54,700行の新規開発の工期(中央値)が約9ヶ月と記載。開発規模(40,000行)の本件は、工期約6.5ヶ月に相当。

■今回事例 (実績値)

開発規模: 約40,000行

開発工期: 71日間 (2.5ヶ月)

要員数 : 2人

※ドメインや複雑度の違い、参考データは中央値である事を考慮して、参考に留める。

3 プロジェクトの主要要素の統計

IPA

表 主要要素の値 (中央値)

項目	改修・保守	新規開発
FP規模	862FP	1,225FP
SLOC規模	22.2KSLOC	54.7KSLOC
工期	6.3ヶ月	9.1ヶ月
工数	7,024人時 (約43.9人月)	12,005人時 (約75.0人月)
月当たりの要員数	6.4人	9.4人

FP (Function Point) : 機能規模量 SLOC (Source Lines Of Code) : ソース行数

© 2018 IPA, Japan

出典:「ソフトウェア開発データ白書20182019」のご紹介
～プロジェクトマネジメントの実践・改善に活かす最新定量データと分析結果～
独立行政法人情報処理推進機構(IPA)社会基盤センター 産業プラットフォーム部 研究員 峯尾正美様

まとめ

まとめ

開発効率を引き出したのは一体なにか？

- 新規開発の場合、派生開発の元となる成果物が無いが、それを(間違っているとしても良いので)敢えて作る事で、結果的に製品について2回以上考える事になる。
- 元となる成果物の設定は、開発の起点を得られれば良く、ある意味での気楽さが、決断の早さと良い課題設定に繋がる。
- トレーサビリティマトリックス(TM)を使って早期に論理矛盾を収束させるため、後工程(実装段階)での試行錯誤を抑制。確信を得た上で変更を加えるため、結果に寄与しない模索を排除。
- 現状からの差分で考えるから、大きく失敗しない。

まとめ

リスクの大幅低減、開発負荷の低減と分散に役立つヒント

- 起点: 新規開発対象に近い既存成果物の見つけ方
 - 論理構成で既存成果物を見ること(見た目ではなく)
 - 抽出した論理構成で最終成果物に向かう論理矛盾が無いこと
 - ヒント: 社内用デバッグツールの機能がユーザーの欲しいモノかも
- 開発: 要求とアーキテクチャと実装の一貫性確保
 - トレーサビリティマトリックス(TM)による検証
 - 抽象インターフェースと実装の関係性意味理解
- 全体: 手段ではなく目的に向かう取り組み
 - 開発効率が上がった分をユーザーへの提供価値増大に向ける
 - 単に「開発効率が上がった」では競争に勝てない

結論

- 派生開発の「機能追加や修正に必要な部分を理解し、変更仕様を定義して、影響範囲を明確化する手順と手法」は、リスクの大幅低減、開発負荷の低減と分散に役立つ。
- 新規開発対象に近い成果物を変更前の既存成果物として定義、新規開発を派生開発として扱う事で、効率的な開発が可能。

時間的な余裕、気持ちの余裕が生まれ、新しいアイデアの創発に繋がる。



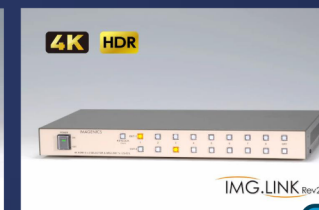
提供価値の増大 / 開発チームの自信



デバイスの自動探索

Windows, macOSのクロスプラットフォーム対応

マニュアルなしで使える



HDMI信号に対応した映像信号スイッチャー

業務用システムに必要な各種機能をコンパクトに搭載した高品質な製品群

拘り抜いた品位の高い挙動と抜群の安定性でクリティカルな現場に最適