

# 探索的テストを対象とする機械学習(SOM)を利用した 進行中プロジェクトにおける探索箇所推測手法 「FaRSeT-#/ ファルセットシャープ」の提案

## A Proposal of “FaRSeT-#” as the Inferring Method of Exploring Points using Self-Organizing Maps for Exploratory Testing in the Progressing Project

○喜多 義弘<sup>1)</sup>  
○Yoshihiro Kita<sup>1)</sup>

上田 和樹<sup>2)</sup>  
Kazuki Ueda<sup>2)</sup>

櫻井 清敬<sup>2)</sup>  
Kiyotaka Sakurai<sup>2)</sup>

**Abstract** It is a problem that the rework of exploratory testing is occurred by change specifications of software. We proposed the supporting method as named “FaRSeT” in SQiP Symposium 2018, in order to solve the problem. However, our method has a problem that testers is hard to infer the next exploratory points. In this paper, we propose the method as new named “FaRSeT-#” for inferring the exploring points. Our method uses self-organizing maps and k-means method for dividing test sessions into some clusters. It can infer the next exploring points by analyzing the sessions which are included each cluster.

### 1. はじめに

ソフトウェア開発において、開発途中での製品の仕様変更に伴うテストの手戻りが多く発生している。このことは、開発から納品までの工期が短い「短納期型開発プロジェクト」でも同様であり、短い工期中に発生するテストの手戻りは大きな負担になる問題がある。我々はこの問題の解決策として、FaRSeT (Flexible and Rapid Software Test)<sup>[1]</sup>というテスト手法を提案し、SQiP シンポジウム 2018 にて発表した。

FaRSeT を用いた探索的テストにより、事前にテストケースを準備する必要がなくなり、仕様変更が多い部分の手戻りを解消することができた。また、重要な探索箇所についてステークホルダの合意を得ながら優先して探索的テストを実施できるようになり、テストの効率化を図ることもできた。

しかしながら、以下 2 点の問題点が浮き彫りとなった。

- 未実施のテストは不具合が発生していないため、重要な探索箇所として判断できない。
- 不具合数だけでは、残存不具合を想定することができない。

これらの問題点により、重要な探索箇所を判断するための根拠が欠けるため、ステークホルダに探索箇所の合意を得る際の障害になっている。

そこで本研究では、これらの問題点を解決するために、重要な探索箇所を推測し、それを提示する手法「FaRSeT-# (ファルセットシャープ)」を提案する。重要な探索箇所を判断するための手法として、機械学習の 1 つである自己組織化マップ (Self-Organizing Maps)<sup>[2][3]</sup>を利用し、セッションを二次元マップ上に可視化し提示する。

---

1) 長崎県立大学

University of Nagasaki

〒851-2195 長崎県西彼杵郡長与町まなび野 1-1-1 長崎県立大学シーボルト校

Tel: 095-813-5107 E-mail: kita@sun.ac.jp

University of Nagasaki Siebold Campus, Manabino 1-1-1, Nagayo, Nishi-sonogi, Nagasaki, Japan

2) 日本ナレッジ株式会社

Nihon Knowledge Co., Ltd.

【キーワード：】探索的テスト FaRSeT 自己組織化マップ 探索箇所推測

## 2. 関連研究

### 2.1. FaRSeT

FaRSeT (Flexible and Rapid Software Test) <sup>[1]</sup>は、マインドマップによる業務分析と探索的テストマトリクスを使用し、進行中のプロジェクトにおけるテスト探索個所の優先度を判断することで、柔軟かつ迅速にテストを行う手法である。FaRSeT の手順を以下に示す。

- ① マインドマップを活用した業務分析を実施する。
- ② 手順 1 の業務分析をもとに、品質特性からブレークダウンしたテストチャータ（探索的テストの抽象目的）を作成する。
- ③ テストチャータを横軸、機能を縦軸とした「探索的テストマトリクス」（図 1 参照）を作成する。
- ④ 探索的テストマトリクスにおける各項目の交点を「セッション（探索的テストの実施範囲）」と定義し、重要度が高いセッションをステークホルダの合意を得ながら、探索的テストを実行する。
- ⑤ 手順④のテストにて不具合を発見した場合は、探索的テストマトリクスの当該セッション箇所にも不具合数を記載する。
- ⑥ セッションに記載した不具合数をもとに、次の探索的テストを行うセッションを決定しつつ、手順④から手順⑤を繰り返す。

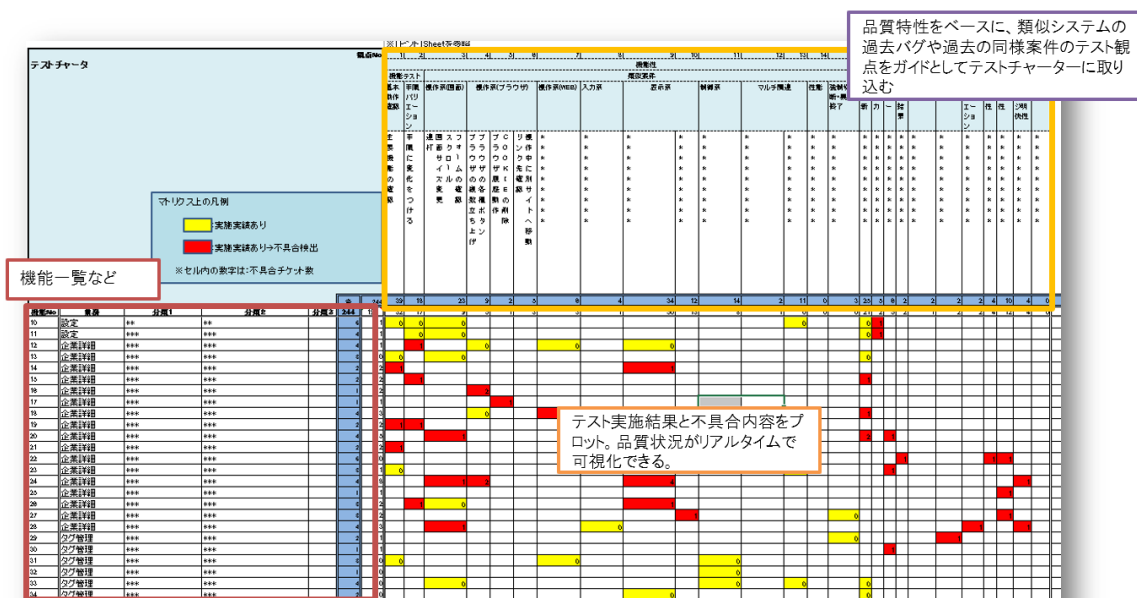


図 1 探索的テストマトリクスの例

手順⑥においては、テストチャータと合わせて機能全体を俯瞰したり、特定の機能に絞ったりしながら不具合が多い箇所を探索する。例えば図 2 の探索的テストマトリクスにおいて、縦軸の機能では機能 A と機能 I の不具合数が多く、一方、横軸のテストチャータ（この例では品質特性）では、機能完全性、機能正確性、運用操作性の不具合数が多い。

これらにより、図中の色づいたセッションが重要で優先すべき探索箇所として判断することができる。色のついていないセッションにおいても、機能やテストチャータの内容によって、担当者がその重要度を判断し、探索的テストを実行する。

### 2.2. 自己組織化マップ

自己組織化マップ (Self-Organizing Maps, 以下, SOM とする) <sup>[2][3]</sup>とは、教師なしの競合学習型ニューラルネットワークの 1 つであり、与えられた多次元データの類似度を二次元マップ上に表現する機械学習モデルである (図 3 参照)。SOM は、入力層と競合層の 2 つの層から成り、入力層には入力ベクトルを持つノードを、競合層には参照ベクトルを持つノードをそれぞれ含む。

機能	1	2	3	4	5	6	7	8	9	10	11	12
	機能完全性	機能正確性	機能適切性	共存性	相互運用性	適切度認識性	習得性	運用操作性	ユーザエラー防止性	ユーザインターフェース快楽性	アクセシビリティ	機密性
A	10	12	9	0	1	1	0	4	5	2	0	0
B	0	0	0	0	0	0	0	0	0	0	0	0
C	5	6	1	0	0	3	0	3	1	0	0	0
D	4	9	1	0	0	0	0	2	0	1	0	0
E	2	7	2	0	0	0	0	2	0	0	1	0
F	3	8	2	0	0	0	0	1	0	3	0	0
G	2	3	2	2	1	1	0	4	1	1	1	0
H	0	3	0	0	0	1	0	0	1	0	0	0
I	9	23	4	0	2	5	0	6	1	2	0	0
J	1	2	0	0	0	0	0	3	3	0	0	0
K	1	2	0	1	0	0	0	0	0	0	0	0

図 2 探索的テストマトリクスにおける重要な探索箇所

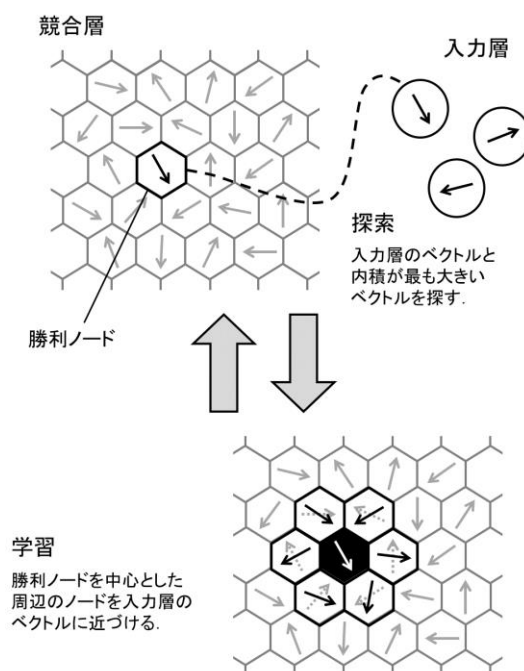


図 3 SOM の概要

入力層のノードは、入力として用いるデータの標本数と同じ数だけあり、それぞれのノードにつきベクトル化した標本を入力ベクトルとして保持している。一方、競合層のノードは格子状または六角形ハニカム状の配置になっており、初期状態にはランダムな参照ベクトルを保持している。SOM における学習手順を以下に示す。

- (1) 競合層のすべての参照ベクトルに対して 1 つの入力ベクトルを比較する。
- (2) 入力ベクトルに最も類似した参照ベクトルを持つノードを「勝利ノード」とする。
- (3) 勝利ノードからある一定の距離内にある参照ベクトル（勝利ノードが持つ参照ベクトルも含む）に対し、入力ベクトルにさらに類似させるべく、ベクトルの加減を行う。手順(1)～(3)を「学習」と呼び、全ての入力ベクトルで行うことにより学習 1 回分とする。
- (4) 既定の学習回数を満たすまで、学習を繰り返す。
- (5) 既定回数の学習が完了した後、再び勝利ノードを算出し、その勝利ノード上に該当する入力層を配置する。

これにより、入力したデータの標本のうち、類似している標本同士は互いに距離が近い位置に入力層が配置され、類似していない標本は距離が遠い位置に入力層が配置される。そして、標本間の類似具合が入力層同士の距離となって二次元空間上で可視化することができる。

### 2.3. k-means 法

k-means 法<sup>[4]</sup>とは、平面上に散在するデータを互いの距離によって k 個のグループに分ける非階層型クラスタリング（教師なし機械学習）の 1 つである。このグループのことをクラスと呼び、k は任意の値である。k-means 法のアルゴリズムを以下に示す。

- (1) 各データをランダムにクラスに割り当てる。
- (2) 同じクラスに属するデータ間の距離に対して、クラスの重心を求める。
- (3) 各データのクラスを、最寄りの重心のクラスに割り当て直す。
- (4) 全データにおいてクラスの変化がなくなるまで、手順(2)および(3)を繰り返す。

k-means 法を用いて、SOM 上に配置された各入力層を k 個のクラスに機械的に分ける。そして、クラスごとに入力層を分析することで、各クラスの傾向を導き出すことができる。

### 3. 探索箇所推測手法「FaRSeT-#」の提案

本提案手法は、既存手法である「FaRSeT」に機械学習である SOM を取り入れて拡張した形になる。探索的テストマトリクスから各セッションをデータの標本として SOM に入力し、学習させる。SOM の特徴を活かし、セッションを性質の類似具合でマッピングし、k-means を用いてクラスタリングを行い、セッションをクラスタにまとめる。これにより、同じクラスタに属するセッションは類似していると捉えることができ、不具合の傾向、探索の狙いどころ、および有効なテスト手法が共通していると考えられる。そして、各クラスタに含まれるセッションを分析し、探索が必要な箇所や不具合が潜在する箇所を推測する。

図4は、図2の探索的テストマトリクスをSOMで学習した結果を、二次元マップにマッピングしたものである。マップ中の各点におけるラベルは、探索的テストマトリクスでの交点となる行と列の組み合わせでつけており、例えば、ラベル「A5」は、探索的テストマトリクスの行「機能A」と列「テストチャータ5（この場合は相互運用性）」を表す。

SOM に入力するデータとして、以下の項目から成るセッション情報を対象とする。

- 機能に関するメトリクス  
例) 機能の規模、開発担当者のスキルなど
- テストチャータに関するメトリクス  
例) 品質特性の重要度など
- セッションに関するメトリクス  
例) 不具合数、直近のテスト結果、テスト実施時間、テスト担当者のスキルなど

表 1 に色別クラスタにおけるセッション個数とクラスタ内の類似度を示す. 本提案では,  $\cos$  類似度を用いてクラスタ内のセッション間の類似度を算出し, 全てのクラスタで偏ることなく, 類似度が最も高くなるクラスタ数  $k$  を選ぶ. 表 1 より, 類似度の標準偏差は 2.2% 程度に収まっていることから,  $k=6$  が妥当であると判断し, 6 つのクラスタに分類する. 各クラスタを 6 色 (RED, BLUE, GREEN, YELLOW, CYAN, MAGENTA) に配色する. 図 4 より, マップ上の点 (セッション) が各色に分かれており, 6 つのクラスタが形成されていることが確認できる.

表 1 色別クラスタにおけるセッション個数とクラスタ内類似度

	RED	BLUE	GREEN	YELLOW	CYAN	MAGENTA
セッション個数	35	22	33	19	12	11
クラスタ内類似度	79.78%	78.26%	79.86%	77.91%	74.57%	74.50%

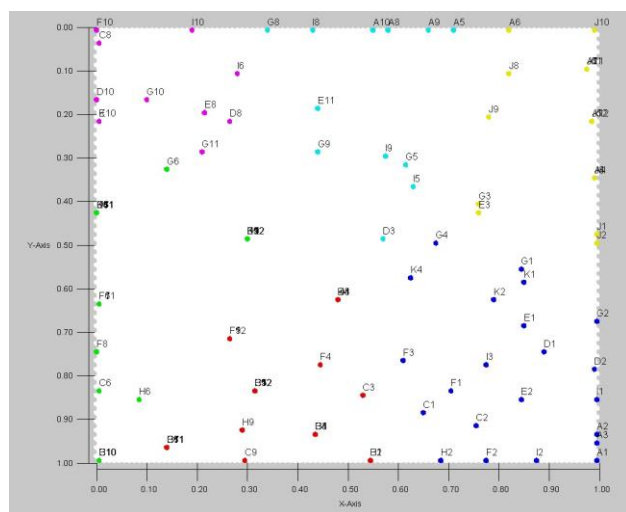


図 4 図 2 の探索的テストマトリクスを表現した SOM の例

次に、クラスタごとにセッションを分析し、不具合の傾向を導き出す。図 5～図 8 に、図 2 の探索的テストマトリクスを例とし、特徴ごとにセッションの分布具合を着色した二次元マップを示す。なお、着色した特徴と色については、図 5 では機能規模を青で、図 6 では品質特性重要度を緑で、図 7 および図 8 では不具合数および重要不具合を赤でそれぞれ示し、その色が濃いほどその特徴が大きく表れている。

図5より, 青く着色された部分には YELLOW, 次いでBLUEやCYANのクラスタが集まっている. 図6より, 緑に着色された部分には RED やBLUEのクラスタが集まっている. 図7より, 赤く着色された右下の部分には BLUE, 薄く着色された上の部分には YELLOW, CYAN, MAGENTA のクラスタが集まっている.



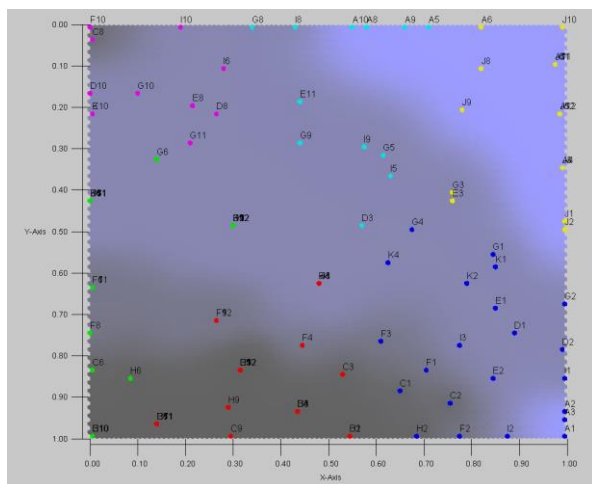


図 5 機能規模の分布

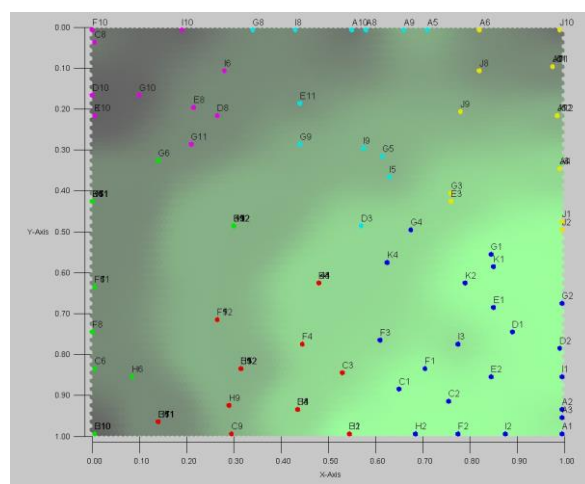


図 6 品質特性重要度の分布

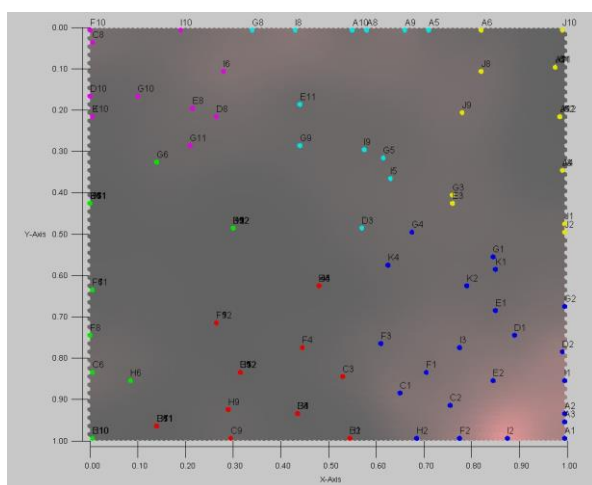


図 7 不具合数の分布

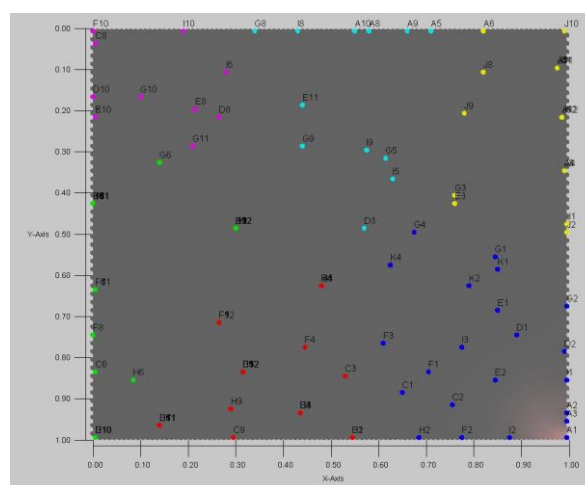


図 8 重要不具合の分布

図 8 では、赤く着色された部分は右下に集まり、BLUE のクラスが該当している。これらのマップから、以下のことが推測できる。

- BLUE は、全ての特徴に大きく関わり、かつ重要不具合が出ていることから、優先的に探索を実施すべき箇所である。
- 不具合は YELLOW, CYAN, MAGENTA から出ているが、YELLOW は機能規模が大きいため、今後も不具合が発見される可能性が高く、探索を実施すべき箇所である。
- RED は、不具合が出ていないものの、品質特性重要度が高いため、探索箇所として考慮すべきである。
- GREEN は、不具合が少なく、各特徴において関わるのが小さいため、探索の優先度は低い。

この例では、探索的テストをテストの熟練者 2 名、中級者 1 名、初心者 1 名の計 4 名で行っている。各位が発見した不具合の分布を、図 9 および図 10 に示す。これらの図を図 7 および図 8 と併せることにより、熟練者は重要な不具合を中心とする多くの不具合を的確に摘出していることを確認できる。一方、中級者は不具合を的確に摘出しているものの範囲は小さく、初心者は左上の MAGENTA にあたる「重要な品質特性ではない（図 6 参照）」箇所での不具合を摘出していることを確認できる。これらをまとめると、以下の推測ができる。

- 中級者や初心者は BLUE だけでなく、CYAN や MAGENTA から不具合を発見している。
- CYAN や MAGENTA は機能規模や品質特性重要度は小さいため、熟練者だけでなく中級者や初心者も担当可能である。

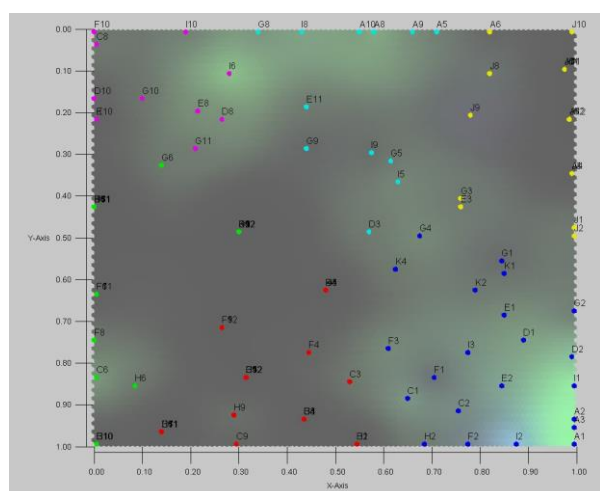


図 9 熟練者 2 名（青と緑）が発見した不具合の分布

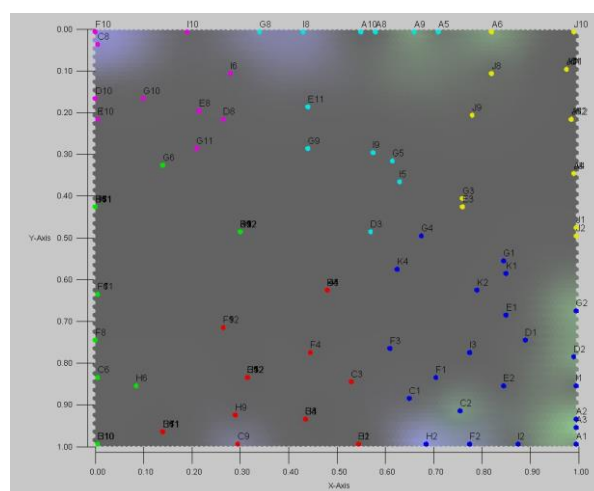


図 10 初心者（青）と中級者（緑）が発見した不具合の分布

SOM を用いることにより、不具合の傾向が視覚的に分かりやすくなった。表 2 に、色別の分けたクラスタごとのメトリクス集計結果を示す。表 2 から、前述した全ての推測に相違がないことが確認できる。これらをもとに探索の優先順位は、BLUE, YELLOW, RED, 同列で CYAN および MAGENTA, GREEN の順である。そして、BLUE および YELLOW は熟練の担当者が優先して実施すべきであり、CYAN および MAGENTA は中級者や初心者でも担当可能である。

表 2 クラスタごとのメトリクス集計結果

			1セッションあたりの不具合数平均								平均度数	
			不具合重要度別			テスト実施担当者別				全体		
クラスタ	セッション 個数	不具合数	高	中	低	担当 A (熟練者)	担当 B (熟練者)	担当 C (中級者)	担当 D (初心者)		機能規模	品質特性 重要度
RED	35	3	0.00	0.09	0.00	0.06	0.00	0.03	0.00	0.09	1.34	3.23
BLUE	22	127	0.05	5.36	0.36	2.68	2.68	0.14	0.27	5.77	2.86	4.77
GREEN	33	6	0.00	0.18	0.00	0.18	0.00	0.00	0.00	0.18	2.58	2.24
YELLOW	19	14	0.00	0.68	0.05	0.26	0.42	0.00	0.05	0.74	4.79	3.00
CYAN	12	29	0.00	1.67	0.75	1.58	0.50	0.17	0.08	2.33	3.67	2.58
MAGENTA	11	20	0.00	1.64	0.18	1.27	0.27	0.27	0.00	1.82	2.73	1.45
全クラスタ	132	199	0.01	1.60	0.22	1.01	0.65	0.10	0.07	1.82	2.99	2.88

図 11 に図 2 をクラスタごとに色分けした図を示し、この図をもとに、図 12 には探索箇所の優先度を、図 13 には次なる探索箇所を、図 14 には中級者や初心者が担当可能である箇所をそれぞれ示す。これらの図により、次なる探索箇所や各担当者が担当すべき箇所を把握することができる。また、従来手法では図 2 しか示すことができず探索箇所の根拠が十分ではなかったが、本提案手法では、機能規模、品質特性重要度、不具合数をもとに算出しているため、これらの要素を根拠として示すことができる。

#### 4. 妥当性の評価

本提案手法の妥当性を確認するために、実際に進行中である開発プロジェクトに適用した。方法として、 $n$  回目 ( $n$  は任意の数) の探索的テストを実施後、その結果を本提案手法に適用した場合と、従来どおりに  $n+1$  回目の探索的テストを行い、検出した不具合について分析し、比較する。表 3 には探索的テスト  $n$  回目におけるメトリクス集計結果を示す。この結果は、提案手法によるクラスタ分けを行い、クラスタごとに集計している。なお、表中の色がついた項目は、SOM で学

機能	1	2	3	4	5	6	7	8	9	10	11	12
	機能完全性	機能正確性	機能適切性	共存性	相互運用性	適切認識性	習得性	運用操作性	ユーザエラー防止性	ユーザ/FF快楽性	アクセシビリティ	機密性
A	10	12	9	0	1	1	0	4	5	2	0	0
B	0	0	0	0	0	0	0	0	0	0	0	0
C	5	6	1	0	0	3	0	3	1	0	0	0
D	4	9	1	0	0	0	0	2	0	1	0	0
E	2	7	2	0	0	0	0	2	0	0	1	0
F	3	8	2	0	0	0	0	1	0	3	0	0
G	2	3	2	2	1	1	0	4	1	1	1	0
H	0	3	0	0	0	1	0	0	1	0	0	0
I	9	23	4	0	2	5	0	6	1	2	0	0
J	1	2	0	0	0	0	0	3	3	0	0	0
K	1	2	0	1	0	0	0	0	0	0	0	0

図 11 クラスタごとの色分け

機能	1	2	3	4	5	6	7	8	9	10	11	12
	機能完全性	機能正確性	機能適切性	共存性	相互運用性	適切認識性	習得性	運用操作性	ユーザエラー防止性	ユーザ/FF快楽性	アクセシビリティ	機密性
A	10	12	9	0	1	1	0	4	5	2	0	0
B	0	0	0	0	0	0	0	0	0	0	0	0
C	5	6	1	0	0	3	0	3	1	0	0	0
D	4	9	1	0	0	0	0	2	0	1	0	0
E	2	7	2	0	0	0	0	2	0	0	1	0
F	3	8	2	0	0	0	0	1	0	3	0	0
G	2	3	2	2	1	1	0	4	1	1	1	0
H	0	3	0	0	0	1	0	0	1	0	0	0
I	9	23	4	0	2	5	0	6	1	2	0	0
J	1	2	0	0	0	0	0	3	3	0	0	0
K	1	2	0	1	0	0	0	0	0	0	0	0

図 12 探索箇所 の優先度  
(赤色の濃い順)

機能	1	2	3	4	5	6	7	8	9	10	11	12
	機能完全性	機能正確性	機能適切性	共存性	相互運用性	適切認識性	習得性	運用操作性	ユーザエラー防止性	ユーザ/FF快楽性	アクセシビリティ	機密性
A	10	12	9	0	1	1	0	4	5	2	0	0
B	0	0	0	0	0	0	0	0	0	0	0	0
C	5	6	1	0	0	3	0	3	1	0	0	0
D	4	9	1	0	0	0	0	2	0	1	0	0
E	2	7	2	0	0	0	0	2	0	0	1	0
F	3	8	2	0	0	0	0	1	0	3	0	0
G	2	3	2	2	1	1	0	4	1	1	1	0
H	0	3	0	0	0	1	0	0	1	0	0	0
I	9	23	4	0	2	5	0	6	1	2	0	0
J	1	2	0	0	0	0	0	3	3	0	0	0
K	1	2	0	1	0	0	0	0	0	0	0	0

図 13 次なる探索箇所  
(BLUE および YELLOW)

機能	1	2	3	4	5	6	7	8	9	10	11	12
	機能完全性	機能正確性	機能適切性	共存性	相互運用性	適切認識性	習得性	運用操作性	ユーザエラー防止性	ユーザ/FF快楽性	アクセシビリティ	機密性
A	10	12	9	0	1	1	0	4	5	2	0	0
B	0	0	0	0	0	0	0	0	0	0	0	0
C	5	6	1	0	0	3	0	3	1	0	0	0
D	4	9	1	0	0	0	0	2	0	1	0	0
E	2	7	2	0	0	0	0	2	0	0	1	0
F	3	8	2	0	0	0	0	1	0	3	0	0
G	2	3	2	2	1	1	0	4	1	1	1	0
H	0	3	0	0	0	1	0	0	1	0	0	0
I	9	23	4	0	2	5	0	6	1	2	0	0
J	1	2	0	0	0	0	0	3	3	0	0	0
K	1	2	0	1	0	0	0	0	0	0	0	0

図 14 中級者や初心者が担当可能である箇所  
(CYAN および MAGENTA)

習対象にした項目である。表 4 には探索的テスト n+1 回目を実施した際のメトリクス集計結果を示す。

表 3 より, GREEN は不具合数および重要不具合件数の発見数が最も多いことから優先的にテストする箇所であると言え, MAGENTA は不具合数が 2 番目に多く, その不具合はテストスキル C (初心者) の担当者によって発見されていることから, 中級者以下でも担当できる箇所であると言える。一方, RED は不具合数が 3 番目に多いが, 重要不具合件数が出ていないことから, それらの不具合は重要ではないと判断でき, RED に対してテスト工数をかけなくてよいと推測できる。

BLUE は不具合数が MAGENTA より少ないが, テスト工数が全クラスタ中で最も多く, YELLOW はテスト工数をかけているものの, 不具合が全く出ていない。一方, CYAN では全くテストを行



表 3 探索的テスト n 回目におけるクラスタごとのメトリクス集計結果

クラスタラベル	平均：不具合数	平均：テストスキル A の発見数	平均：テストスキル B の発見数	平均：テストスキル C の発見数	平均：機能重要性 (3>2>1)	平均：機能規模 (3>2>1)	平均：観点優先度 (3>2>1)	平均：新規要件に関連 (1 or 0)	平均：重要不具合件数 (Critical)	平均：重要不具合件数 (Major)	平均：重要不具合件数 (Minor)	平均：修正件数 (Fixed)	平均：工数/h
RED	1.5	1.4	0.1	0.0	2.8	2.8	2.4	0.9	0.0	0.0	0.0	0.5	2.1
BLUE	0.9	0.4	0.4	0.1	2.8	2.6	2.4	0.8	0.2	0.2	0.2	0.3	16.8
GREEN	3.0	1.6	1.4	0.1	2.9	2.9	2.3	0.9	1.7	1.7	0.0	1.9	9.1
YELLOW	0.0	0.0	0.0	0.0	2.5	2.7	2.0	0.8	0.0	0.0	0.0	0.0	2.3
CYAN	0.0	0.0	0.0	0.0	2.6	2.6	0.9	0.8	0.0	0.0	0.0	0.0	0.0
MAGENTA	1.7	0.3	0.2	1.2	2.9	2.6	2.3	0.8	0.7	0.7	0.0	0.8	4.7

表 4 探索的テスト n+1 回目を実施した際のメトリクス集計結果

クラスタラベル	平均：不具合数	平均：テストスキル A の発見数	平均：テストスキル B の発見数	平均：テストスキル C の発見数	平均：重要不具合件数 (Critical)	平均：重要不具合件数 (Major)	平均：重要不具合件数 (Minor)	平均：修正件数 (Fixed)	平均：工数/h	時間当たりの検出件数
RED	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.94	0.00
BLUE	0.06	0.00	0.06	0.00	0.00	0.06	0.00	0.00	4.00	0.02
GREEN	0.50	0.25	0.25	0.00	0.25	0.25	0.00	0.08	5.65	0.09
YELLOW	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.15	0.00
CYAN	0.09	0.00	0.09	0.00	0.09	0.00	0.00	0.00	2.43	0.04
MAGENTA	0.11	0.00	0.11	0.00	0.11	0.00	0.00	0.00	3.72	0.03

っており、その理由として、観点優先度が低いことが表より推測できる。これらから、BLUE や YELLOW では不具合がこれ以上出ない、もしくはテスト工数に見合うだけの不具合は出ないことが推測でき、その分のテスト工数を CYAN に回した方が不具合を発見しやすいと予想する。

上記の推論を踏まえて表 4 を確認すると、GREEN や MAGENTA から多くの不具合が検出され、MAGENTA はテストスキル B (中級者) でも発見できていることから推測どおりであった。RED, BLUE, YELLOW についても、テスト工数の割に不具合が出ておらず。一方、CYAN は BLUE よりもテスト工数は少ないが、不具合が多く出ているため、これらの点についても推測どおりであった。

## 5. おわりに

本研究では、探索的テストにおいて、重要な探索箇所の判断や残存不具合の想定ができないという問題点を解決するために、重要な探索箇所を推測し、それを提示する手法「FaRSeT-#」を提案した。SOM や k-means 法を用いて機械的にマッピングやクラスタリングを行うことで、客観的に探索箇所を判断でき、その根拠を明示することができた。また、テストの担当者ごとにマップを分析することで、担当すべき箇所を担当者の能力に合わせて決定することもできる。

## 参考文献

- [1] 上田和樹, 丹場順次, 工藤修悟, “短納期型開発プロジェクトのためのテスト手法「FaRSeT (Flexible and Rapid Software Test)」の適用と効果”, ソフトウェア品質シンポジウム 2018, pp.1-8, 2018.
- [2] T. Kohonen, “Self-Organizing Maps”, Springer-Verlag, 2001.
- [3] T. Kohonen 著, 徳高平蔵, 大蔵又茂, 堀尾恵一, 藤村喜久郎, 大北正昭 監修, “自己組織化マップ” 改訂版, Springer-Verlag, 2005.
- [4] D. Steinley, M.J. Brusco, “Initializing k-means Batch Clustering: A Critical Evaluation of Several Techniques”, Journal of Classification, Vol.24, No.1, pp.99-121, 2007.