規模と複雑さおよび変更に着目した組込みソフトウェアアーキテクチャの保守性評価事例

鷲崎 弘宜[†1]　津田 直彦[†1]　溝口 将史[†2]
加藤 有真[†2]　吉村 健太郎[†2]

概要：組込みソフトウェアの開発にあたり，ハードウェアや環境，要求の変化に応じた修正や変更，拡張に向けて高い保守性をアーキテクチャのレベルで備えることが望ましい．我々は一つの車載ソフトウェア開発プロジェクトを対象に，アーキテクチャを構成するハードウェアへの依存性や役割に応じたレイヤの単位で，規模と複雑さおよび変更に着目した保守性の測定および評価を試みた．評価の結果，レイヤ別で規模や複雑さの傾向が幾らか異なり，それに応じて変更のされやすさが幾らか異なる可能性を明らかとした．本評価事例により得られた知見は，他の組込みソフトウェア開発における保守性の評価と改善へと役立てられる可能性がある．

キーワード：メトリクス，ソフトウェア保守，ソフトウェアアーキテクチャ，車載ソフトウェア

A Case Study on Automotive Software Architecture Maintainability Evaluation by focusing on Size, Complexity and Modifications

Hironori Washizaki[†1]　Naohiko Tsuda[†1]　Masashi Mizoguchi[†2]
Yuma Kato[†2]　Kentarou Yoshimura[†2]

1. はじめに

特定用途の計算機システムに組み込まれて用いられる組込みソフトウェアの開発にあたり,ハードウェアや環境,あるいは要求の変化に応じた変更や誤りの修正，さらにはプロダクトラインに代表されるシリーズ展開といった保守を効率よく実施できるための高い保守性を，アーキテクチャのレベルで備えることが望ましい．

ソフトウェアの保守性評価の取り組みは様々にあるが，組込みソフトウェアのアーキテクチャ上の特性を考慮した保守性評価の実際の報告は限定的である．

組込みソフトウェア全般の保守性向上に資する技術として特に組込みソフトウェアアーキテクチャの特性に着目し，プロダクトメトリクスやプロセスメトリクスに基づく評価方法を検討した．検討に基づき，特に規模，複雑さおよび変更に着目した保守性測定を実際の車載ソフトウェアに適用し，測定結果に基づいてソフトウェアアーキテクチャの保守性の評価を試みた．評価の結果，レイヤ別で規模や複雑さの傾向が幾らか異なり，それに応じて変更のされやすさが幾らか異なる可能性を明らかとした．本評価事例により得られた知見は，他の組込みソフトウェア開発における保守性の評価と改善へと役立てられる可能性がある．

本稿では以降において，2 節では組込みソフトウェアのプロダクトおよびプロセスの両面から保守性評価の方法とプロセスを検討する．3 節では，事例研究として実際の車載ソフトウェアを対象に測定および評価した結果を示し考察する．4 節で本稿をまとめる．

2. 組込みソフトウェアにおける保守性評価

我々は，ハードウェア依存性や役割に応じたアーキテクチャ上のレイヤに着目し，保守に関係するプロダクト上の特徴とプロセス上の特徴の両面から保守性を測定し分析評価することを検討した．以下において検討したレイヤおよび測定の方法を説明する．

2.1 アーキテクチャ上のレイヤ

組込みソフトウェアにおいてはアーキテクチャを階層的に構成し，役割に応じてソフトウェアをモジュール化することで，多様なハードウェアへと対応および再利用可能とすることが求められる．本稿で取り上げる典型的な組込みソフトウェアのアーキテクチャとして，車載ソフトウェアアーキテクチャ標準 AUTOSAR Classic Platform [1] を例に構成を図 1 に示し，主要なレイヤの役割を以下に示す．

- アプリケーションソフトウェア（Application Software: ASW）：最上位のレイヤにあたり，ハードウェアに依存せず，自動車システムの制御機能を実現するソフトウェアコンポーネント群から構成される．
- ランタイム環境（Run Time Environment: RTE）：アプリケーションソフトウェア内のコンポーネント間，および，アプリケーションソフトウェアと基盤ソフトウェア間をつなぐアプリケーションインタフェースである．
- 基盤ソフトウェア（Basic Software: BSW）: OS やミド

†1 早稲田大学 Waseda University
†2 日立製作所 Hitachi, Ltd.

ルウェアに相当し，マイクロコンピュータの抽象化やエンジンコントロールユニット（ECU）の抽象化といったハードウェアとソフトウェアをつなぐ役割を担う．

上述のようにレイヤによって役割およびハードウェアへの依存性が大きく異なる．従って，プログラムを構成するモジュールやファイルの規模や複雑さ，変更のされやすさも大きく異なると推測される．

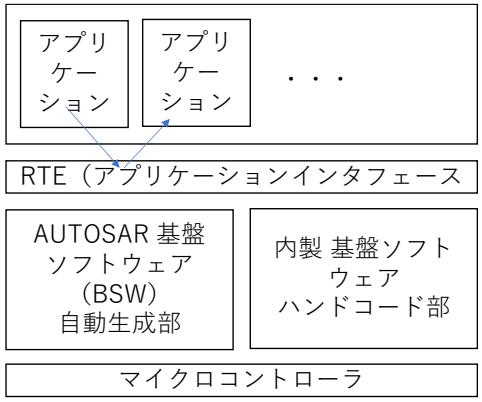

図 1. AUTOSAR CP のレイヤ構成

2.2 規模と複雑さ

C 言語で記述されたプログラムソースコードの各ファイルについて，代表的な規模および複雑さに関するプロダクトメトリクスを取り上げる．種別とメトリクスの概要を表 1 に示す．これはすべて，既存の静的解析ツール（たとえば Understand [2]）を用いて容易に測定可能である．

これらの規模や複雑さが極端に大きい場合に，当該ファイルの修正や変更は容易ではなく，それらのファイル群から構成されるアーキテクチャ上のレイヤもまた保守性が低いと考えられる．

表 1. 規模および複雑さのメトリクス（一部）

種別	説明
関数	ファイル内の関数定義数
行数	物理行数
行数	コード行数
行数	実行行数
行数	プリプロセッサ行数.
行数	コメント行数（およびコメント比率）
行数	空白行数
複雑度	サイクロマティック複雑度最大値
複雑度	制御構造（if, while, for, switch など）の最大ネストレベル値
結合度	関数の入力数（利用グローバル変数の数＋利用引数の数＋呼び出す側の関数の数）の最大値
結合度	関数の出力数（代入/変更グローバルオブジェクト数＋代入/変更引数の数＋呼び出し関数の数＋戻り値が void 以外の場合の当該関数カウント）の最大値

2.3 変更回数

ファイルの修正や変更，拡張といった保守活動の直接的な実績として，プロセスメトリクスとしてのファイルの変更回数を取り上げる．

プロダクトメトリクスの規模や複雑さが極端に大きい場合に，当該ファイルの変更は容易ではなく，要求の変更や追加に応じた拡張や適応のための保守活動にあたり，より多くの変更を必要とする可能性がある．

また，複雑さの極端な大きさはしばしば欠陥の含まれやすさと関係があることが知られており，結果として欠陥修正のための変更を多く必要とする可能性がある．

なお後述の事例研究においては，変更の理由によらず，欠陥修正のための変更とそれ以外の変更（機能追加や拡張，適応などのための変更）を区別せず変更回数をカウントしている．

2.4 測定と分析評価のプロセス

測定と分析評価に有用なプロセスとしては，ISO/IEC 25040:2011 [3] をはじめとして，Experience Factory (Quality Improvement Paradigm) [4] およびそれと関わる GQM [5] や GQM+Strategies [6] におけるものなど様々にある．ここでは，将来的な組織展開を念頭に，Experience Factory や GQM+Strategies におけるプロセスをベースに次のように設計した．大枠を図 2 に示す．

1. 特性化: 文脈や環境を明らかとし，適用範囲を設定する．特に実証研究においてレイヤへの着目から，開発運用の経緯や環境が明らかな対象ソフトウェアの選定と構成レイヤ群から複数の異なるレイヤを選択する．
2. 目標設定: 組込みソフトウェアアーキテクチャの特にレイヤ構成に着目し，レイヤ別の規模，複雑さおよび変更回数の測定を通じて保守性を明らかとすることを目標とし，研究課題として明確化する．
3. 評価プロセス定義: 研究課題に回答するために，対象ソフトウェアの開発運用を手掛ける開発チームと，測定評価の経験に長けた研究チームの連携による測定評価の流れを明確とする．分析評価の詳細を以下とする．
   ① フォルダの単位で全体の規模・複雑さおよび変更回数の分布傾向を把握する．
   ② 当該レイヤの役割に照らして保守の観点からその適切さや今後の改善必要性を考察する．
   ③ 分布において標準から著しく逸脱し特異な値を

示すファイル群を特定し，その内容および今後の保守性向上に向けた改善の要否を考察する．

4. 評価プロセス実行: 開発チームと研究チームが連携して対象や目標を設定し，対象の特性に基づいて研究チームが検討した測定評価の方法（特にメトリクスおよび測定ツール）を両チームで検討および合意し，開発チームが測定した結果を研究チームが分析評価し，評価案を両チームで検討および合意する．
5. 結果の分析: 規模・複雑さと変更回数の関係を，特異な値を示すファイル群に絞った傾向や，全体の相関などから分析する．
6. 全体の改善: レイヤ別に保守性の現状および将来に向けた改善可能性を考察するとともに，他の類似組込みソフトウェアの開発・運用における保守性評価と改善に適用可能な知見としてまとめる．そのうえで，規模・複雑さの改善を通じた将来における変更回数の提言可能性を考察し，対象ソフトウェア以外へも展開可能とすることを目指す．加えて，測定評価プロセスそのものの改善をあわせて検討する．

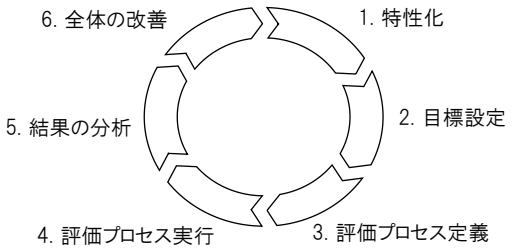

図 2. 測定評価のプロセス

3. 事例研究

実際の車載ソフトウェアを対象に，上述の測定評価プロセスを実施した．測定評価の対象と結果および保守性評価のうえでの考察を以下に示す．

3.1 対象ソフトウェア

特性化にあたり，同一の車載ソフトウェア開発プロジェクトにおける表 2 に示す二つのシステム Sb，Sa を対象とした．

表 2. レイヤ別の規模

項目	Sb	Sa
レイヤ	基盤ソフトウェア	アプリケーションソフトウェア
フォルダ	ルート，サブフォルダ 4 個 Sb1～Sb4	ルート，サブフォルダ 20 個 Sa1～Sa20
C 言語ソースファイル	600 個弱	1,500 個強
関数	1,500 個強	2,100 個強

3.2 研究課題

目標設定にあたり，研究課題を以下に設定した．

- **RQ1. 組込みソフトウェアのアーキテクチャ上の異なる各レイヤにおいて規模や複雑さはどのようになっているか？**
  この研究課題に回答するため，レイヤ別に保守に影響する可能性のあるプロダクトメトリクスとしてプログラムのファイル当たりの規模や複雑さを測定し，その要因や将来の保守に与えうる影響を考察した．
- **RQ2. 組込みソフトウェアのアーキテクチャ上のレイヤの違いに応じて変更されやすさは異なるか？**
  この研究課題に回答するため，レイヤ別に保守上の直接的なプロセスメトリクスとしてファイル当たりの変更回数を測定し，レイヤ別の違いを比較評価および考察した．
- **RQ3. 規模・複雑さと変更されやすさにはレイヤごとにどのような関係があるか？**
  この研究課題に回答するため，レイヤごとに特異な変更回数示すファイルに着目して規模や複雑さとの関係を調べるとともに，レイヤごとに全体的な相関を調査した．

3.3 規模と複雑さ

定義した評価プロセスを実行し，規模と複雑さについて以下の結果を得た．この結果により RQ1 に回答する．

(1) Sb の測定

コード行数の測定結果をフォルダ別に図 3 に示す．大半のフォルダについてファイルの行数は 100 行以下であるが，一つのフォルダ Sb2 について 2,000 行と特異な値を示す三つのファイルを特定した．それらのファイルについて複雑さを確認したところ，サイクロマティック複雑度に代表される複雑さは小さなものであり，複雑な分岐処理の箇所ではないことを確認した．そこで，これらのファイルに処理を集中させることの意図や妥当性を確認することが望ましい可能性を考察した．プリプロセッサ行数の観点から，数百行になるヘッダーファイルが Sb2 および Sb4 に複数存在することを確認した．これは#ifdef 分岐や#define の数が多いためであり，機種展開などを見越して可変としている要素数の多さが，コードの保守性の低下へとつながっている可能性を考察した．

規模に加えて複雑さの観点から全体を分析した結果，規模から見た場合と同様に Sb2 および Sb4 について，サイクロマティック複雑度最大値が 10 以上のファイルを多数確認した（参考 図 4）．それらのファイルの幾つかについては制御構造の最大ネストレベル値や関数の入力数も大きいことを確認した．それらの複雑なファイルについて，ごく少数の関数で複雑なネスト処理をしていることが原因と見受けられた．それらのファイルについて，今後の変更が不要と推測されるものではなく，機種展開にあたり機種ごと

の編集が多く必要となる個所であればネストを分解・リファクタリングして拡張性および可読性を上げることが望ましい可能性を考察した．

以上より，Sb について全体的には規模および複雑さが概ね抑えられているものの，一部について規模および複雑さが特異に大きなファイルを特定した．その主な要因は，基盤ソフトウェアとして様々なハードウェアや機種へと展開する上で必要となりうる可変性を設けたためと推測できた．複雑さを一部の箇所に局所化できている点は保守の観点から優れていると評価できる一方で，将来における機種展開等が実際になされない場合は技術的負債となっていると捉えられるため，その要否を継続検討する必要性を確認した．

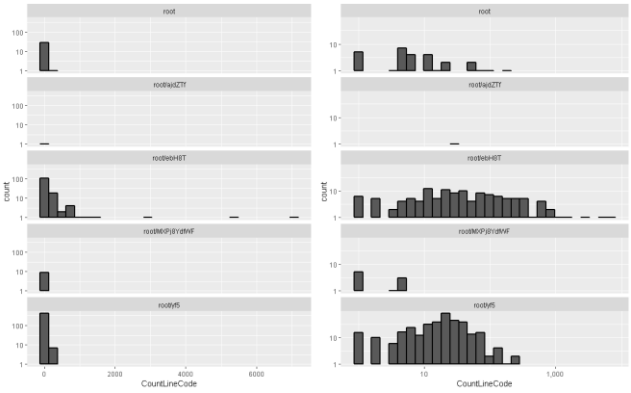

図 3. Sb のコード行数のフォルダ単位のヒストグラム（左側: X 軸線形・Y 軸対数，右側: XY 軸共に対数，最上段が Sb ルートフォルダ，以降は順に Sb1～Sb4，図中のタイトル表記は匿名化後のもの）

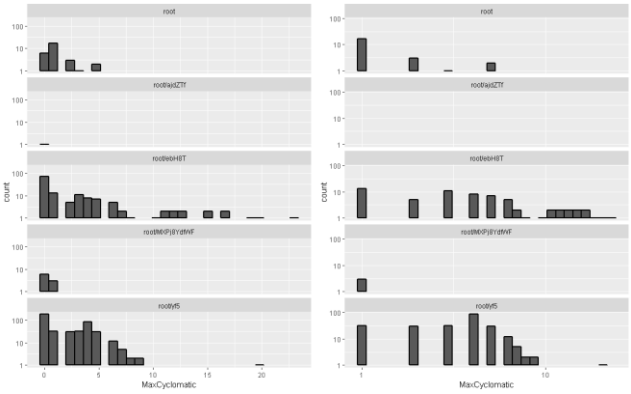

図 4. Sb のサイクロマティック複雑度最大値のフォルダ単位のヒストグラム（左側: X 軸線形・Y 軸対数，右側: XY 軸共に対数）

(2) Sa の測定

コード行数の測定結果をフォルダ別に図 5 に示す．大半のファイルで実行コード行数が 1000 行以下であるが，四つのフォルダ Sa4, Sa8, Sa9, Sa10, Sa12 では，コード行数が 1000 行を越えるファイルが存在し，それらについて一つの例外を除いて全体的に規模に加えて複雑度や結合度も大きい傾向を確認した．担う機能量が多すぎる可能性があり，開発都合上の理由も含めて検討余地の可能性を確認した．なお例外としては，規模が大きく関数定義数も多いものの複雑さが低いファイルが存在し，その定義の妥当性を念のため確認することが望ましいことを考察した．

他の規模の観点として 500 行以上のプリプロセッサ行を持つファイルが三つのフォルダに存在し，条件分岐や条件判定，あるいは可変させる要素の数が多いことに伴い，コードの保守性の低下へとつながっている可能性を考察した．

複雑さについて，四つのフォルダにおいてサイクロマティック複雑度最大値が 30 以上のファイルがあり，一部は 100 以上に達する特異なものがあることを確認した（参考図 6）．一方で，制御構造の最大ネストレベル値について 5 を越えるファイルがいくつか存在するが 10 を越えるものは無く，ネストの深さよりも条件分岐の連続や分岐数の多い switch-case により制御フローの複雑度が上がっていると考えられることを考察した．これらの複雑なファイルについて将来的な機能追加などに応じて変更が必要な場合は拡張性や可読性向上のために関数の分解やリファクタリングが必要な可能性を考察した．

以上より，Sa について全体的には規模および複雑さが概ね抑えられているものの，規模および複雑さが特異に大きなファイルが Sb に比較して複数のフォルダに存在することを確認した．将来的な機種展開その他の理由により変更の可能性がある場合はそれらの分解やリファクタリングを通じた規模・複雑さ集中の低減の要否を継続検討する必要性を確認した．

RQ1. 組込みソフトウェアのアーキテクチャ上の異なる各レイヤにおいて規模や複雑さはどのようになっているか？

基盤ソフトウェアとアプリケーションソフトウェアのいずれも全体的には規模および複雑さが概ね抑えられているものの，一部に特異に大きなファイルを特定した．基盤ソフトウェアでは特異なファイルは限定され，様々なハードウェアや機種へと展開する上で必要な変化点を局所的に設けているためと推測できた．一方でアプリケーションソフトウェアにおいては，特異に大きなファイルが複数のフォルダに存在し，将来的な拡張や機能追加に伴い変更の可能性がある場合はそれらの分解やリファクタリングを通じた規模・複雑さ集中の低減の要否を継続検討する必要性を確認した．

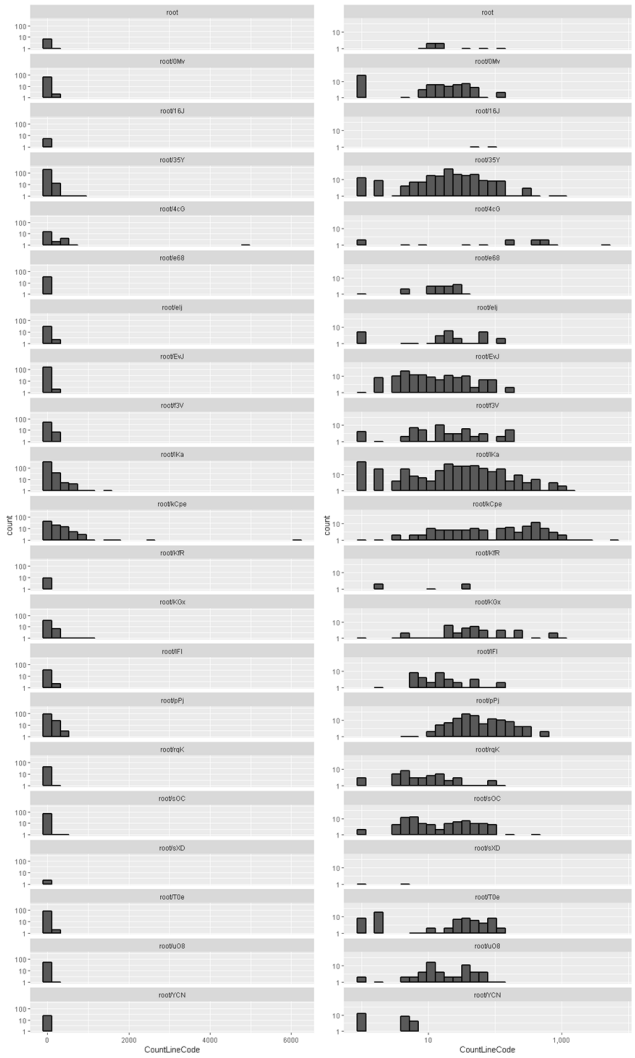

図 5. Sa のコード行数のフォルダ単位のヒストグラム（左側: X 軸線形・Y 軸対数，右側: XY 軸共に対数，最上段が Sa ルートフォルダ，以降は順に Sa1～Sa20，図中のタイトル表記は匿名化後のもの）

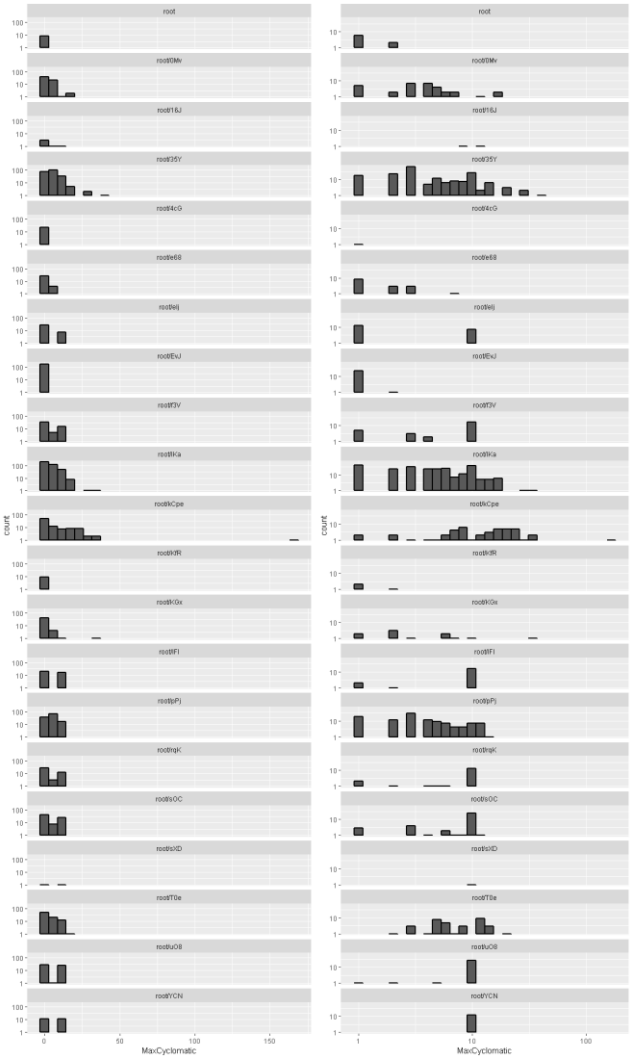

図 6. Sa のサイクロマティック複雑度最大値のフォルダ単位のヒストグラム（左側: X 軸線形・Y 軸対数，右側: XY 軸共に対数）

3.4 変更回数

定義した評価プロセスの実行により変更回数について図 7 に示す結果を得た．この結果により RQ2 に回答する．

両システムについて変更回数が 10~20 の付近にややかたまって該当するファイル群があることを確認した．そのうえでシステム間の比較において，Sa のほうがファイルの変更回数が多い傾向を確認した．マンホイットニーの U 検定の結果，p 値が十分に小さく，両システムにおいて統計的に有意な変更回数の差があることを確認した．ただし差の大きさを判断するための効果量として Cliff のデルタを分析した結果，分布の差の大きさは小さいと判定された．

具体的には，Sa において 2 回以上変更しているファイルが半数を占め，さらには変更回数が 50 や 100 を越えるファイルが複数あることを確認した．その要因として，幅広いファイル群の変更が同時に必要な修正や要求・環境変化対応の必要性が数多くあったことを推測し，将来においても同様の変更が必要となる可能性を考察した．

主として機能追加や拡張に伴う対応を，基盤ソフトウェアよりもアプリケーションソフトウェアにおいて多く対応していることが要因と推測され，安定性に応じたレイヤ構成の観点からは好ましい可能性を考察した．

RQ2. 組込みソフトウェアのアーキテクチャ上のレイヤの違いに応じて変更されやすさは異なるか？
基盤ソフトウェアとアプリケーションソフトウェアのいずれにおいても変更回数が 10~20 の付近にややかたまって該当するファイル群があることを確認した．そのうえで，アプリケーションソフトウェアのほうがファイルの変更回数が多い傾向を確認した．これは主として機能追加や拡張に伴う対応を，基盤ソフトウェアよりもアプリケーションソフトウェアにおいて多く対応していることが要因と推測され，安定性に応じたレイヤ構成の観点からは好ましい可能性を考察した．

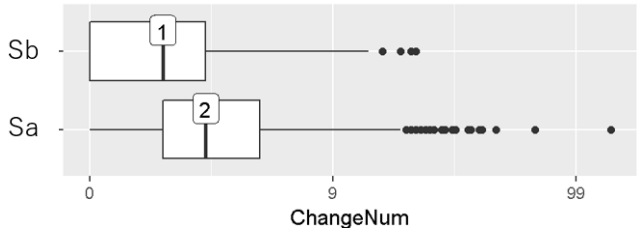

図 7. 変更回数（横軸が対数目盛，測定値ゼロの場合を扱うため対数変換前に+1）

3.5 規模・複雑さと変更の関係

結果の分析として，ファイル変更回数について特異な値を示すファイル群に着目し，それらの規模・複雑さを把握することで，規模・複雑さと変更の関係を分析した．この結果により RQ3 に回答する．

具体的には，システム毎にファイルの変更回数が上位 95%以上となるファイルを特異点群として区別し，他は通常群とした．そのうえでシステム毎に特異点群と通常群の規模・複雑さの分布を比較した．測定結果を図 8 に示す．両システムとも変更回数の特異点群はコメント行が多い傾向を確認した．これは，当該ファイルが扱う処理や関心事の複雑さや難易度の高さに応じて，その説明のためにコメントが多くなり，かつ，修正や見直しのために頻繁な変更を生じている可能性を考察した．

また，両システムでは変更頻度の多いファイルについて他の規模・複雑さの傾向が幾らか異なることを確認した．具体的には，Sb の特異点群において複雑度の大きい傾向にあり，一方で，実行部分の行数の差は小さい可能性があることを確認した（参考 図 9，図 10）．逆に Sa の特異点群において実行部分の行数が多い傾向にあり，一方で複雑度の差は小さい可能性があることを確認した．Sb は基盤ソフトウェアであることから，特定ハードウェアに依存し，元々複雑な処理の箇所のパラメータ変更をすることが多い可能性を考察した．また Sa はアプリケーションソフトウェアであることから，変更要因の多くは機能追加であり，変更が必要な箇所であるほど機能規模が累積していく可能性を考察した．Sa の特異点群は宣言行を多用しているためその命名やスコープの適切さの再確認，ならびに，関数に対する出力数が多いため，その適切さの再確認の可能性を考察した．

続いて，規模・複雑さおよび変更回数の相関関係を両システムについて分析し，上述の特異群に着目した分析に概ね沿って，システムによって相関の傾向が異なることを確認した．求めた相関係数のうちで，ファイルの変更回数に関連する相関係数のみ抜粋したものを表 3 に示す．

具体的には Sb について，変更回数とコード行数，コメント行数，サイクロマティック複雑度最大値および制御構造の最大ネストレベル値との正相関が比較的大きいことを確認した．

Sa について変更回数とコード行数，コメント行数との正相関が比較的大きいことを，Sb と共通に確認した．一方で Sb と異なる点として，Sa において行数中のコメントの割合の負相関を確認した．これは，コメントでないコード行数が多いという状況を反映したものの可能性がある．加えて，関数の出力数の最大値との正相関が比較的大きいことを確認した．変更回数の多いファイルでは，外部変数や関数の参照が多い可能性がある．

RQ3. 規模・複雑さと変更されやすさにはレイヤごとにどのような関係があるか？
基盤ソフトウェアとアプリケーションソフトウェアのいずれにおいても変更回数の特異点群はコメント行が多い傾向を確認した．一方で，変更頻度の多いファイルの複雑度および実行行数に傾向の違いを確認した．基盤ソフトウェアにおいては特定ハードウェアに依存し，元々複雑な処理の箇所のパラメータ変更をすることが多い可能性を考察した．アプリケーションソフトウェアにおいては変更要因の多くは機能追加であり，変更が必要な箇所であるほど機能規模が累積していく可能性を考察した．特に，宣言行における命名やスコープ，外部変数や参照に起因する関数の出力の多さの適切さの再確認の可能性を考察した．

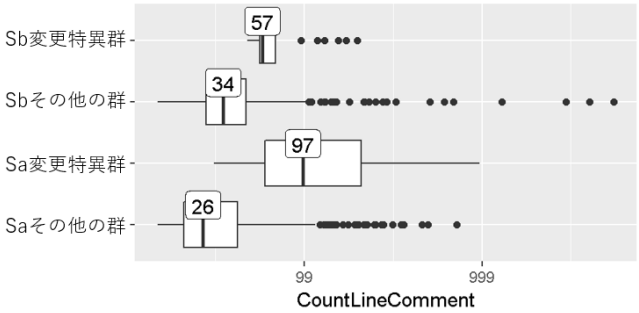

図 8. コメント行数（横軸が対数目盛）

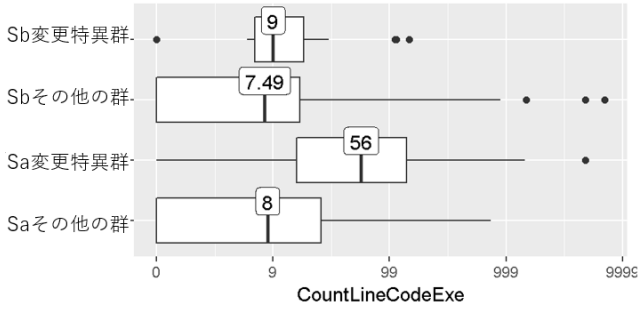

図 9. 実行行数（横軸が対数目盛）

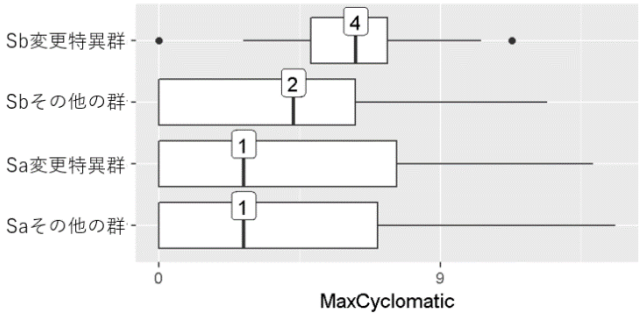

図 10. サイクロマティック複雑度最大値（横軸が対数目盛）

表 3. ファイルの変更回数に対するスピアマンの順位相関係数（絶対値が 0.4 以上のものを強調表示）

メトリクス	Sb	Sa
ファイル内の関数定義数	0.21	**0.59**
物理行数	0.33	**0.63**
コード行数	0.27	**0.61**
実行行数	0.22	**0.62**
プリプロセッサ行数	0.15	-0.28
コメント行数	**0.40**	**0.69**
コメント比率	-0.02	-0.37
空白行数	0.14	0.38
サイクロマティック複雑度最大値	0.31	0.11
最大ネストレベル値	0.35	0.09
関数の入力数の最大値	0.21	0.07
関数の出力数の最大値	0.10	0.39

3.6 知見のまとめ

全体の改善として，測定評価により得られた知見をまとめ，対象ソフトウェア以外についても展開可能とすることを目指す．具体的には，規模，複雑さおよび変更とそれらの間の関係について，レイヤ別に確認した事項と将来の保守性向上に向けた施策の可能性を表 4 にまとめた．これらは，当該システムのみならず当該ソフトウェアを構成する他のシステムへの展開，さらには，他ソフトウェアへと，レイヤ構成の類似性を確認することで援用できる可能性がある．

表 4. レイヤ別の特徴と保守性向上に向けた考察

観点	基盤ソフトウェア Sb	アプリケーションソフトウェア Sa
規模および複雑さ	特異なファイルは限定され，様々なハードウェアや機種へと展開する上で必要な可変性を局所的に設けているためと推測．	特異に大きなファイルが複数のフォルダに存在し，将来的な拡張や機能追加に伴い変更の可能性がある場合はそれらの分解やリファクタリングを通じた規模・複雑さ集中の低減の要否を継続検討する必要性．
変更	変更回数が 10~20 の付近にややかたまって該当するファイル群があることを確認した．全体的にアプリケーションソフトウェアよりも変更は抑制的．安定性に応じたレイヤ構成の観点からは好ましい可能性．	ファイルの変更回数が多い傾向を確認．主として機能追加や拡張に伴う対応を，基盤ソフトウェアよりもアプリケーションソフトウェアにおいて多く対応していることが要因と推測．
総合的	特に変更の多いファイルへの着目～，特定ハードウェアに依存し，元々複雑な処理の箇所のパラメータ変更をすることが多い可能性．その設計の妥当性や機種展開を見据えた可変性について継続確認．	特異に大きなファイルが複数のフォルダに存在し，将来的な拡張や機能追加に伴い変更の可能性がある場合はそれらの分解やリファクタリングを通じた規模・複雑さ集中の低減の要否を継続検討する必要性．

4. おわりに

本稿では組込みソフトウェア全般の保守性向上に資す

る技術として特に組込みソフトウェアアーキテクチャの特性に着目し，プロダクトメトリクスやプロセスメトリクスに基づく評価方法とプロセスを検討した．検討に基づき，特に規模，複雑さおよび変更に着目した保守性測定を実際の車載ソフトウェアに適用し，測定結果に基づいてソフトウェアアーキテクチャの保守性の評価を試みた．

評価の結果，レイヤ別で規模や複雑さの傾向が幾らか異なり，それに応じて変更のされやすさが幾らか異なる可能性を明らかとした．本評価事例により得られた知見は，他の組込みソフトウェア開発における保守性の評価と改善へと役立てられる可能性がある．

今後は，得られた評価方法，プロセスおよび知見を，当該ソフトウェアを構成する他のシステムへと展開し，レイヤの特性に応じた規模，複雑さ，変更および関連の特徴への影響をさらに分析する予定である．加えて，レイヤ構成の類似性を持つ他の組込みシステムへと一部適用することで，プロセスや知見の一般性を確認し，さらなる取り組み全体の改訂を進める予定である．また，組込みソフトウェアにおける再利用性をはじめとする他の品質特性や [7][8], C 言語プログラムにおけるアーキテクチャ上の他の特徴 [9][10] との関係も分析調査することを検討している．

参考文献

[1] Simon Fürst, Markus Bechter, “AUTOSAR for Connected and Autonomous Vehicles: The AUTOSAR Adaptive Platform,” Proceedings of the 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W), pp. 215-217, 2016

[2] Scientific Toolworks, Understand, https://www.scitools.com/

[3] ISO/IEC 25040:2011 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Evaluation process, 2011

[4] Victor Basili, “The Experience Factory and Its Relationship to Other Quality Approaches,” Advances in Computers, Vol. 41, pp. 65-82, 1995

[5] Victor Basili and H. Dieter Rombach, “Goal, Question, Metric Paradigm,” Encyclopedia of Software Engineering, Vol. 1, pp. 528-532, 1994

[6] Victor Basili, Adam Trendowicz, Martin Kowalczyk, Jens Heidrich, Carolyn Seaman, Jurgen Munch, Dieter Rombach 著, 鷲崎弘宜, 小堀貴信, 新谷勝利, 松岡秀樹 監訳, 早稲田大学グローバルソフトウェアエンジニアリング研究所ゴール指向経営研究会訳, “ゴール&ストラテジ入門: 残念なシステムの無くし方 (GQM+Strategies)”, オーム社, 2015

[7] Hironori Washizaki, Yasuhide Kobayashi, Hiroyuki Watanabe, Eiji Nakajima, Yuji Hagiwara, Kenji Hiranabe, and Kazuya Fukuda, “Experiments on Quality Evaluation of Embedded Software in Japan Robot Software Design Contest,” Proceedings of the 28th IEEE/ACM International Conference on Software Engineering (ICSE 2006), pp.551-560, 2006

[8] Hironori Washizaki, Toshikazu Koike, Rieko Namiki, Hiroyuki Tanabe, “Reusability Metrics for Program Source Code Written in C Language and Their Evaluation,” Proceedings of the 13th International Conference on Product-Focused Software Development and Process Improvement (Profes 2012), pp.89-103, 2012

[9] Devansh Tiwari, Hironori Washizaki, Yoshiaki Fukazawa, Tomoyuki Fukuoka, Junji Tamaki, Nobuhiro Hosotani and Munetaka Kohama, “Metrics driven architectural analysis using dependency graphs for C language projects,” Proceedings of the 43rd IEEE Computer Society Signature Conference on Computers, Software and Applications (COMPSAC 2019), pp. 117-122, 2019

[10] Devansh Tiwari, Hironori Washizaki, Yoshiaki Fukazawa, Tomoyuki Fukuoka, Junji Tamaki, Nobuhiro Hosotani, Munetaka Kohama, Yann-Gael Gueheneuc, “Commit–Defect and Architectural Metrics–based Quality Assessment of C language,” Proceedings of the 15th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2020), pp. 579-586, 2020