学士論文

2025年度

論文題目

ソフトウェアプロジェクトの時系列変化の可視化によるプロジェクト理解支援

京都産業大学 情報理工学部

学生証番号 154217

氏　　名 速水 健杜

概要

ソフトウェア開発の現場において，GitHub 上でのプロジェクト管理は標準的な手法となっている．しかし，大規模なプロジェクトでは，コードベースの複雑さが増大し，開発者やプロジェクトマネージャーがプロジェクトの全体像を把握することが困難になっている．この課題に対応するため，本研究では新しい動的チャート作成ツール「RepoTimelapse」を提案する．

RepoTimelapse は，プロジェクトの構造を可視化する TreeMap と，時系列での変化を表現する動的棒グラフを組み合わせることで，プロジェクトの多面的な理解を支援する．この手法の特徴は，3D 表現を避け 2D ベースの可視化を採用することで，処理負荷を軽減しながら，誰もが理解しやすい表現方法を実現している点にある．都市などの複雑なメタファーに依存せず，一般的なグラフ表現を用いることで，技術的背景や経験レベルに関わらず，幅広いユーザーが容易にプロジェクトの全体像を把握できるようになってる．

本研究では，GitPython を用いたバージョン管理情報の収集と Plotly による可視化の組み合わせにより，プロジェクトの時系列的な変化を効率的に収集・分析・表示する機能を実装する．評価実験では，異なる規模と特性を持つ複数の GitHub プロジェクトを対象に，処理性能と可視化品質の両面から検証を行う．その結果，大規模プロジェクトにおいても実用的な処理時間での動作が確認され，提案手法の有効性が示された．

今後の展望として，多言語プロジェクトにおける分析機能の強化，大規模プロジェクトでのパフォーマンス最適化，そしてユーザーフィードバックに基づく UI の改善と機能拡張を計画している．本研究は，ソフトウェア開発プロセスの理解と改善に新たな視点をもたらし，プロジェクト管理の効率化に貢献することを目指す．

謝辞

本稿は以下の方々なくして，存在し得なかったでしょう．研究に関する助言や資料の提供などご指導くださった玉田春昭教授．そして研究室のみなさん．心より感謝しております．

目 次

図 目 次

表 目 次

第1章　はじめに

近年のソフトウェア開発において，システムの大規模化・複雑化は著しい進展を見せており，その開発・保守作業はこれまで以上に困難を極めている状況にある．特に，プロジェクトの規模が拡大するにつれて，コードベースの複雑性は指数関数的に増大し，開発者やプロジェクト管理者が全体の構造や進捗状況を正確に把握することが極めて困難になっている．このような開発チームの全体像把握の困難さは，重複した開発作業や非効率なリソース配分，さらには将来的な技術的負債の蓄積といった深刻な問題を引き起こす可能性が高い．

現代のソフトウェア開発において，GitHub は最も広く普及しているプラットフォームとしての地位を確立している．その直感的な使いやすさと強力な協調機能により，個人開発者から大規模企業に至るまで，極めて幅広いユーザー層に積極的に採用されている．つまり，ソフトウェア開発に関する膨大なデータの大部分が GitHub 上に集積されており，これらの貴重なデータを効果的に活用することで，ソフトウェア開発プロセスの大幅な効率化や品質向上が期待できる状況にある．

このような背景を踏まえ，本研究では，GitHub 上に蓄積された豊富なデータを効果的に活用し，ソフトウェアプロジェクトの革新的な可視化手法の確立を目指している．この可視化アプローチでは，複雑なメタファーを用いることなく，誰もが直感的に理解できる一般的なグラフ形式を採用することを基本方針としている．本研究では，TreeMap と動的棒グラフという 2 種類の革新的な動的グラフを効果的に組み合わせることで，プロジェクトの構造的側面と成長的側面の両面を統合的に可視化する手法を提案する．この革新的なアプローチにより，プロジェクトの構造把握と成長過程の分析が格段に容易になり，開発者やプロジェクト管理者の意思決定を強力かつ効果的に支援することが期待される．

第2章　関連研究

本研究に関連する先行研究として，以下の 3 つの観点から関連研究を整理する．

1. CodeCity 関連の研究
2. Information Seeking 支援に関する研究
3. メトリクスの可視化に関する研究

2.1　CodeCity 関連の研究

CodeCity は，ソフトウェアシステムを都市メタファーを用いて 3D で可視化するツールである [1]．このツールは，クラスを建物として表現し，パッケージを地区として表現する．その際，メソッド数を建物の高さに，属性数を建物の底面積にマッピングし，コードの行数を建物の色の濃さでマッピングする特徴を持つ．これにより，ソフトウェアの構造を直感的に理解することを目指している．また，都市の属性と実際のソフトウェアメトリクスを関連付けることで，複雑なソフトウェアシステムの全体像を把握しやすくする工夫がなされている．

しかし，CodeCity には複数の課題が存在する．第一に，3D 表現による複雑性と高い処理負荷の問題がある．特に大規模システムでは，3D 空間内の情報が過密になり，個々の要素の識別や関係性の把握が困難になる傾向がある．第二に，都市の属性とメトリクスの対応理解が必要となる点が挙げられる．利用者は建物の面積や高さが何を表しているのかを理解する必要があり，これが学習コストとなっている．第三に，大規模プロジェクトでは表示が複雑化する傾向があることが指摘されている．この問題は，3D 描画に伴う高い処理負荷とともに，CodeCity の実用上の大きな障壁となっている．

2.2　Information Seeking 支援に関する研究

OSS における Information Seeking(IS) は開発時間の約 60%を占める重要なサブタスクである [3]．IS は情報の欠落や点在により発生し，新規参入者がプログラムから必要な情報を見つけ出すコストを高める要因となっている．このような課題に対し，IS を支援するアプローチとして，GitHub の Issue や Pull Request などのソーシャルコーディングツール上の実体とその関連をグラフで可視化する研究が行われている．

これらの研究では，GitHub API を利用して実体の取得と関連付けを行い，Neo4j をグラフデータベースとして使用することで，関連にプロパティを持たせることを可能としている．具体的には，Pull request とプログラムファイルの関連付け，Issue とプログラムファイルの関連付け，プログラムファイル同士の関連付けなどを実現している．このような可視化により，開発者間のコミュニケーションやコード理解の促進が期待できる．しかし，これらの研究はソーシャルな側面に焦点を当てており，ソフトウェアの構造的な理解の支援については十分に検討されていない．

2.3　メトリクスの可視化に関する研究

ソフトウェアメトリクスの可視化に関する研究は，ソフトウェアの品質評価や保守性の向上を目的として数多く行われている [2]．これらの研究では，メトリクスの目的を明確にし，何を測り，どう活用するかを定義するアプローチが提案されている．具体的には，Goal-Question-Metric(GQM) アプローチによる目的に応じたメトリクスの選定や，メトリクスの統計的分析と可視化によるソフトウェア品質の評価が行われている．

また，メトリクス分析ツールによる効率的な測定と分析の自動化も進められており，Eclipse Metrics Plugin や SourceMonitor，CCFinderX など，様々なツールが開発されている．これらのツールは，LOC（コード行数）や CK（Chidamber-Kemerer）メトリクスなど，多様なメトリクスの測定と分析を支援する．しかし，これらのツールの多くは静的な分析に留まっており，プロジェクトの時系列的な変化や成長を視覚的に表現することは考慮されていない．

これらの先行研究に対して本研究では，3D 表現を避け 2D ベースの可視化を採用することで処理負荷を軽減し，メタファーに依存せず一般的なグラフ表現を用いることで直感的な理解を促進する．また，TreeMap

と棒グラフという 2 種類の動的グラフを組み合わせることで，プロジェクトの構造と成長の両面を表現可能とする．TreeMap はプロジェクトの階層構造を効率的に表現し，動的棒グラフはメトリクスの時系列的な変化を分かりやすく示すことができる．これにより，ソフトウェアの静的な構造と動的な成長の両面から，プロジェクトの理解を支援することが可能となる．

第3章　提案手法

本節では，GitHub プロジェクトの構造把握と成長分析のための革新的な可視化ツールを提案する．このツールは，大規模プロジェクトの複雑性に効果的に対応するために設計されており，従来の 3D 表現に伴う課題を克服することを目指している．具体的には，2D 表現を採用することで，情報の過密化や処理負荷の問題を解決し，より直感的で効率的な分析を可能にする．

提案ツールの特筆すべき特徴は，既存のツールと比較して，より一般的で理解しやすいグラフ形式を採用していることである．これにより，技術的背景や経験レベルに関わらず，幅広いユーザーが容易にプロジェクトの全体像を把握できるようになる．例えば，時系列でのプロジェクトの構造の変化や，ファイル単位でのコミット頻度などを，シンプルかつ効果的に可視化することが可能となる．

さらに，このツールは高度なインタラクティブ機能を備えており，ユーザーは必要に応じて詳細情報を得ることができる．これにより，プロジェクトの進捗状況や潜在的な問題点を特定し，効果的な意思決定を支援することが期待される．

本研究で提案するツールは，単なる可視化ツールにとどまらず，GitHub プロジェクトの管理と分析のための包括的なプラットフォームとしての役割を果たすことを目指している．これにより，開発者，プロジェクト管理者，さらには研究者が，大規模なソフトウェアプロジェクトの動態を深く理解し，効果的に管理・分析できるようになることが期待される．

3.1　概要

本研究では，GitHub プロジェクトの構造把握と成長過程の理解を支援するための動的チャート作成ツール RepoTimelapse を提案する．本ツールは，プロジェクトの構造を表現する TreeMap と，メトリクスの変化を

示す動的棒グラフを組み合わせることで，プロジェクトの多面的な理解を可能にする．

RepoTimelapse の処理フローは，大きく 3 つのフェーズで構成される．第一のフェーズでは，対象プロジェクトをローカル環境にクローンし，Git のコミット履歴から時系列データを収集する．この際，各コミットにおけるファイル構造やファイルの内容を取得し，プロジェクトの進化の過程を追跡できるようにする．

第二のフェーズでは，収集したデータからメトリクスを計算する．ファイルサイズ，コード行数，ファイルの種類別の統計情報など，プロジェクトの特徴を表す様々なメトリクスを各時点で算出する．

第三のフェーズでは，これらのデータを視覚化する．TreeMap によってプロジェクトの階層構造を表現し，動的棒グラフによってメトリクスの時系列変化を示す．ファイルサイズ，コード行数，ファイルの種類別の統計情報など，プロジェクトの特徴を表す様々なメトリクスを算出する．第三のフェーズでは，これらのデータを視覚化する．

TreeMap によってプロジェクトの階層構造を表現し，動的棒グラフによってメトリクスの時系列変化を示す．本手法の主な特徴は，プロジェクトの「構造」と「変化」という 2 つの側面を，直感的に理解しやすい形で表現する点にある．従来の CodeCity などの 3D 可視化手法と異なり，2D ベースの表現を採用することで処理負荷を軽減し，大規模プロジェクトでも快適な操作性を実現している．また，都市などのメタファーに依存せず，一般的なグラフ表現を用いることで，利用者の学習コストを低減している．さらに，TreeMap と動的棒グラフの組み合わせにより，プロジェクトの静的な構造と動的な変化の両面を同時に観察することが可能である．

TreeMap はプロジェクトのディレクトリ構造やファイルの分布を効率的に表現し，動的棒グラフは各メトリクスの時間的な推移を分かりやすく示す．これらの視覚化要素は連動して動作し，任意の時点での状態を詳細に分析できる．

3.2　データ収集とメトリクス計算

本研究では，プロジェクトの理解を促進するため，Git リポジトリから GitPython を用いて時系列データを収集し，各時点でのメトリクスを計算する．

データ収集とメトリクス計算のプロセスについて詳細に説明する．まず，対象プロジェクトを Git リポジトリからローカル環境にクローンする．GitPython ライブラリを使用してリポジトリにアクセスし，コミット履歴を取得する．各コミットに対して，前回のコミットとの差分を計算し，変更のあったファイルを特定する．GitPython の diff メソッドを用いることで，ファイルの追加・削除・変更を正確に追跡することができる．

メトリクスの計算では，ファイルの種類毎の総行数に着目する．変更のあったファイルについて，拡張子に基づいてファイルの種類を判別する．プログラミング言語ファイルや，設定ファイル，ドキュメントファイルなどである．各ファイルについて，コミット時点での行数を取得し，ファイル種別ごとにそれらを集計する．GitPython を用いた差分解析により，プロジェクト全体の構造の変化を効率的に調査することができる．

本手法では，差分のあるファイルのみを解析対象とすることで，大規模プロジェクトでも効率的なデータ収集が可能である．また，GitPython の提供する豊富な API を活用することで，コミット間の関係やファイルの変更履歴を詳細に追跡できる．これにより得られたデータは，次節以降で説明する TreeMap と棒グラフの入力として使用され，プロジェクトの成長過程を視覚的に表現する基礎となる．

3.3 TreeMap による構造の可視化

本研究では，プロジェクトのディレクトリ構造とファイル分布を効率的に表現するために，TreeMap を採用する．TreeMap は階層化されたデータの可視化に適した手法であり，限られた表示領域でも効率的にデータを表現できる．本節では，TreeMap による構造可視化の詳細について説明する．

TreeMap では，画面を再帰的に分割することで階層構造を表現する．最上位レベルではプロジェクトのルートディレクトリが画面全体に対応し，その下位階層のディレクトリやファイルが順次領域を分割していく形で表現される．各領域の大きさは，そのディレクトリに含まれるファイルの総行数に比例するように設定する．これにより，プロジェクト内での各ディレクトリやファイルの相対的な規模を直感的に理解することができる．

領域の色はファイルの種類を表現するために用いる．例えば，Java ファイルは青色，設定ファイルは緑色，ドキュメントファイルは黄色という

ように，ファイルの種類に応じて異なる色を割り当てる．このカラーマッピングにより，プロジェクト内でのファイルタイプの分布を視覚的に把握することができる．また，階層の深さは領域の境界線の太さによって表現する．これにより，ディレクトリの入れ子構造を明確に示すことができる．

さらに，時系列での変化を表現するために，TreeMap をアニメーション化する．コミットが進むごとに，ファイルの追加・削除・変更に応じて TreeMap の構造が更新される．領域のサイズは行数の変化に合わせて徐々に拡大・縮小し，新規ファイルの追加は新しい領域のフェードイン，ファイルの削除は該当領域のフェードアウトで表現する．このアニメーションにより，プロジェクトの構造変化を連続的に追跡することができる．

TreeMap の実装では，効率的なレイアウトアルゴリズムを採用している．各階層でのスペース分割は，アスペクト比を考慮しながら行われ，可能な限り正方形に近い形状の領域を生成するよう設計されている．これにより，領域の視認性と情報の伝達効率を高めている．また，インタラクティブな操作も可能で，マウスオーバーによる詳細情報の表示や，特定領域へのズームなどの機能を提供する．

この TreeMap による可視化により，プロジェクトの全体像を失うことなく，その構造的特徴を効果的に把握することができる．次節で説明する動的棒グラフと組み合わせることで，より包括的なプロジェクト理解が可能となる．

3.4　動的棒グラフによるメトリクスの可視化

プロジェクトの成長過程における各メトリクスの変化を分かりやすく表現するために，本研究では動的棒グラフを採用する．動的棒グラフは時系列データの変化を効果的に表現できる手法であり，メトリクスの相対的な大きさの比較や時間的な推移の理解を容易にする．

動的棒グラフでは，ファイルの種類毎の総行数をメトリクスとして採用する．縦軸は行数を表し，横軸には異なるファイルタイプを配置する．各ファイルタイプは異なる色で表現され，TreeMap での色使いと統一することで，両者の対応関係を分かりやすくしている．例えば，Java ファイルは青色の棒，設定ファイルは緑色の棒として表示される．

時系列の変化は，コミットの進行に合わせて棒グラフの高さが変化することで表現される．新しいコミットが行われると，変更のあったファイ

ルタイプの棒グラフが，新しい行数に応じて高さを変更する．この変化はアニメーションで滑らかに表現され，急激な増減や緩やかな変化など，プロジェクトの成長パターンを視覚的に理解することができる．

また，大規模プロジェクトにおいて，ファイルタイプの種類が多い場合を考慮し，表示するメトリクスを制限する機能も実装している．総行数が一定値以上のファイルタイプのみを表示することで，主要なファイルタイプに焦点を当てた分析が可能となる．これにより，プロジェクトの中心的な構成要素の変化を，より明確に観察することができる．

動的棒グラフは，TreeMap と連動して動作する．特定のコミットを選択すると，TreeMap とグラフの両方が該当時点の状態を表示する．これにより，プロジェクトの構造とメトリクスの変化を，同時に観察することが可能となる．この連携機能は，プロジェクトの成長過程をより深く理解する上で重要な役割を果たす．

このように，動的棒グラフによる可視化は，プロジェクトの時間的な変化を分かりやすく表現し，TreeMap と組み合わせることで，より包括的なプロジェクト理解を支援する．

3.5 実装

提案手法の実装では，GitPython によるバージョン管理情報の収集と Plotly による可視化の 2 つの主要な機能を実現している．以下に，その詳細を説明する．

データ収集部では，GitPython ライブラリを用いて Git リポジトリの解析を行う．GitPython を採用した理由は，Python の豊富なエコシステムを活用できることに加え，Git リポジトリの操作を効率的に行えることが挙げられる．リポジトリのコミット履歴を取得し，各コミットでの変更ファイルを特定する．そして，変更のあったファイルについて，ファイルタイプごとの行数を計算する処理を行う．大規模プロジェクトの解析では，差分のあるファイルのみを処理対象とすることで，計算コストを削減している．可視化部では，Plotly ライブラリを用いて TreeMap と動的棒グラフを実装している．Plotly は高度なインタラクティブ性を持つグラフ描画ライブラリであり，複雑なデータの可視化に適している．TreeMap については，Plotly の提供する階層的データ表現機能を活用し，プロジェクトの構造を効率的に可視化している．また，動的棒グラフについては，Plotly のアニメーション機能を用いることで，メトリクスの時系列変化

を滑らかに表現している．これらの可視化要素は，Plotly のインタラクティブ機能により，ズームやホバー時の詳細情報表示などの操作が可能である．

このように，GitPython と Plotly という 2 つの主要なライブラリを組み合わせることで，プロジェクトの時系列的な変化を効率的に収集し，視覚的に表現することを実現している．シンプルな構成ながら，プロジェクトの構造と成長を効果的に可視化することができる．

第4章　評価

4.1　評価対象プロジェクトの選定

本研究では，提案ツールの有効性を多角的に評価するため，異なる特性を持つ複数の GitHub プロジェクトを対象として評価を実施する．評価に際しては，プロジェクトの規模，開発の活発さ，成熟度，言語構成という 4 つの主要な特性に着目し，これらの特性を網羅的に検証できるよう評価対象を選定した．本章では，まず評価対象プロジェクトの選定理由と評価方法について説明し，続いて実際の評価結果について述べる．

4.1.1　選定基準

プロジェクトの選定にあたっては，以下の 4 つの観点から評価基準を設定した．プロジェクトの規模については，コードベースの総行数，ファイル数，ディレクトリ構造の複雑さ，および開発者数を考慮した．開発の活発さは，単位時間あたりのコミット頻度，コントリビューターの活動量，Issue・Pull Request の処理量から判断した．プロジェクトの成熟度については，継続期間，開発フェーズの違い，バージョン管理の特徴を評価基準とした．言語構成に関しては，単一言語か複数言語か，使用言語の多様性，各言語の使用比率を考慮した．

4.1.2　選定プロジェクト

評価対象プロジェクトの選定にあたっては，プロジェクトの規模を考慮した．規模については，コードベースの総行数およびファイル数から，小規模（10 万行未満），中規模（10-50 万行），大規模（50 万行以上）に分類した．これらのプロジェクトを選定した主な理由は以下の通りである．

JSON-java は小規模で単一言語のプロジェクトとして，提案ツールの基本性能を評価する基準として適している．CPython は大規模な長期プ

ロジェクトとして，複雑な構造における可視化の効果を検証するのに適している．

中規模プロジェクトとしては SQLite を選定した．SQLite は長期にわたって安定的に維持されているプロジェクトとして，持続的な保守管理における提案ツールの有用性を評価するのに適している．これらの多様な特性を持つプロジェクトを評価対象とすることで，提案ツールの適用可能性と有効性を幅広く検証することが可能となる．これらの基準に基づき選定した評価対象プロジェクトの特性を表 4.1 に示す．

表 4.1: 評価対象プロジェクトの特性

プロジェクト	規模	活発さ	成熟度	主要言語
JSON-java	小規模 2 万行	低活性 週 5 コミット	中期 7 年	Java 単一言語
SQLite	中規模 30 万行	中活性 週 20 コミット	長期 20 年	C 単一言語
CPython	大規模 150 万行	高活性 週 100 コミット	長期 30 年	C，Python 複数言語

4.2　メトリクスベースの評価指標

4.2.1　性能測定指標

提案システムの性能を評価するため，処理時間から計測を行う．処理時間については，ツールの実行完了までの時間を測定する．この指標により，システムの実用性を評価する．

4.2.2　可視化品質の評価指標

可視化の品質評価では，TreeMap による構造表示と棒グラフによる拡張子ごとの規模調査の 2 つの観点から評価を行う．

4.3 メトリクス評価結果

4.3.1 性能測定結果

提案ツールの性能評価として，各プロジェクトに対する処理時間の測定を行った．測定結果を表 4.2 に示す．

表 4.2: プロジェクト別の実行性能

プロジェクト	実行時間 (秒)	コミット数	ファイル数
JSON-java	10．87	1，289	144
SQLite	309．55	29，303	1，659
CPython	1，030．16	124，798	4，385

測定結果から，実行時間はプロジェクトの規模（コミット数，ファイル数）に応じて増加することが確認できた．最小規模の JSON-java では約 11 秒，中規模の SQLite では約 310 秒，大規模な CPython では約 1，030 秒の処理時間を要した．

コミット数に対する実行時間の関係を見ると，ほぼ線形な増加傾向を示している．これは，提案ツールのスケーラビリティが良好であることを示唆している．例えば，SQLite は JSON-java の約 23 倍のコミット数を持つが，処理時間は約 28 倍となっており，コミット数の増加に対して処理時間が極端に悪化することはない．

一方，ファイル数との関係においても同様の傾向が見られ，ファイル数の増加に対して処理時間が著しく増加することはなかった．これらの結果から，提案ツールは大規模プロジェクトに対しても実用的な処理時間で動作することが確認できた．

4.3.2 可視化品質の評価結果

提案ツールによる可視化の結果を図 4.1 と図 4.2 に示す．図 4.1 は JSON-java プロジェクトの TreeMap 表示である．単一言語での開発を反映し，一貫した色調で表現されている．図 4.2 は棒グラフを用いて拡張子ごとの行数を示している．

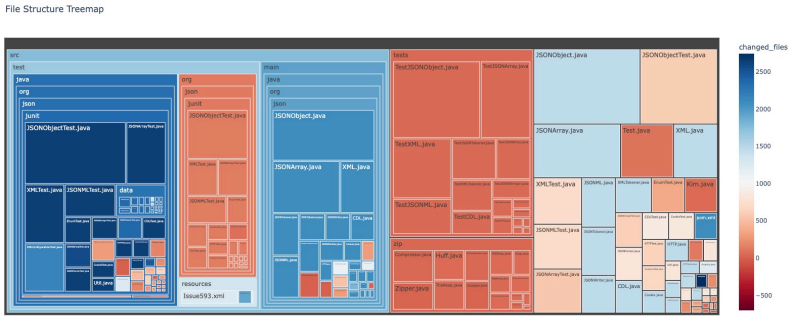

図 4.1: 提案ツールによる可視化結果の例：TreeMap による構造表示

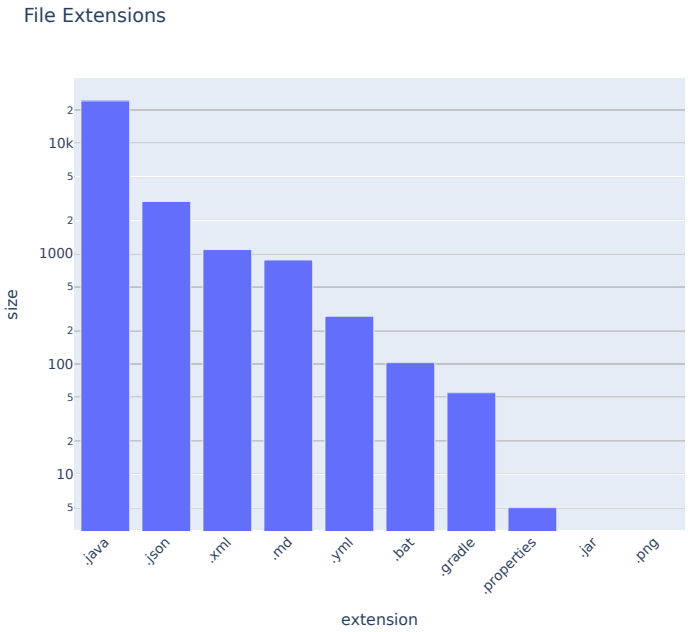

図 4.2: 提案ツールによる可視化結果の例：動的棒グラフによる時系列表示

TreeMap による構造表示では，ディレクトリ階層が明確に表現され，ファイルの種類や規模が直感的に把握できる．また，動的棒グラフでは時系列での変化が滑らかに表現され，プロジェクトの発展過程を効果的に可視化できている．

第5章　考察

評価の結果を通じて，提案ツールの特徴が確認された．スケーラビリティについては，JSON-java から CPython まで，異なる規模のプロジェクトに対して効果的な分析が可能であった．多角的分析では，プロジェクト構造，時系列変化，など，多様な観点からの分析が可能であった．

直感的な可視化については，TreeMap による構造表現や時系列グラフにより，複雑なデータを視覚的に理解しやすい形で提示できた．パフォーマンスに関しては，CPython のような大規模プロジェクトでも，許容可能な処理時間内で分析が完了した．

また，Treemap のみでは，プロジェクトで活発な開発が行われている箇所を特定することが困難な場合がある．各ファイルやディレクトリの大きさは表現できるが，コミット頻度や変更の頻度といった時間的な活性度の情報は直接的に表現することができない．そのため，規模は小さくても頻繁に更新されている重要なモジュールを見落としてしまう可能性が考えられる．

第6章　結論と今後の展望

評価結果により，提案ツールが大規模 GitHub プロジェクトの構造把握と成長分析に有効であることが示された．特に，多角的な分析と直感的な可視化により，プロジェクトの全体像を効果的に把握できることが確認された．

今後の課題としては，より多様なプロジェクト（異なる言語，開発モデル）での検証，言語特有の分析機能の強化，大規模プロジェクトにおけるパフォーマンスの最適化，そしてユーザーフィードバックを基にした UI の改善と機能拡張が挙げられる．これらの課題に取り組むことで，提案ツールの有用性と適用範囲をさらに拡大できると考えられる．

第7章　まとめ

本研究では，GitHub プロジェクトの構造把握と成長分析を目的とした新たな可視化ツールを提案した．本研究で提案した可視化ツールは，GitHub プロジェクトの構造把握と成長分析に新たな視点をもたらし，ソフトウェア開発プロセスの理解と改善に貢献する可能性を示した．今後，上記の課題に取り組むことで，本ツールの有用性と適用範囲をさらに拡大し，ソフトウェア工学分野における重要なリソースとなることが期待される．

関連図書

[1] Richard Wettel, Michele Lanza and Romain Robbes: Software Systems as Cities: A Controlled Experiment, pp. 551–560 (2011).

[2] 阿萬裕久, 野中誠, 水野修: ソフトウェアメトリクスとデータ分析の基礎, コンピュータ ソフトウェア, Vol. 28, No. 3, pp. 3_12–3_28 (2011).

[3] 沓澤脩, 櫨山淳雄ほか: 中大規模 OSS 開発における Information seeking のための可視化ツールの提案, ウィンターワークショップ 2016・イン・逗子 論文集, Vol. 2016, pp. 47–48 (2016).