ChatGPT

TypeScript型エラー分類と型補完・修復に関する研究レビュー

はじめに

TypeScriptを含む静的型付け言語では、コンパイル時に多様なエラーが発生します。特にTypeScriptではエラーにコード（例：`TS2307`, `TS7016`, `TS2339`など）が割り当てられており、エラー種別の特定や頻度分析が容易です。本調査では、コンパイルエラーの分類・分析・定量評価に関する先行研究を整理し、これらの知見が**段階的な型補完・型修復**（エラーコードに応じて段階的に型を補完・修正していく設計）の妥当性を裏付けることを目的とします。具体的には、TypeScriptや他の静的型言語におけるコンパイルエラーの分類研究、CI/IDEでのエラー収集と可視化、自動修復ツールのエラー別対処法、型付け導入によるエラー分布変化の評価、教育的応用、大規模エラーデータセット、エラー指標を用いた品質評価といった観点から10ページ程度の包括的なレビューを行います。

コンパイルエラーの分類に関する研究

コンパイルエラーの**分類（taxonomy）**は各言語や環境で研究されています。TypeScriptでは、エラーコードごとにメッセージが規定されており、それ自体が分類体系と言えます。例えばTypeScriptの一般的なエラーとして、以下のようなカテゴリが知られています [1] [2]:

- **プロパティ不存在エラー（TS2339）**:「プロパティXは型Yに存在しません」というエラーで、存在しないオブジェクトプロパティへアクセスした際に発生します [1]。TypeScript型推論モデルの研究では、このエラーが最も多発するケースが報告されています。例えばYeeらのTypeWeaver評価では、Graph Neural Networkを用いるLambdaNetモデルの出力でTS2339エラーが突出して多く（24,000件以上）発生し、これは**オブジェクト型を具体的に推測する分、誤ったプロパティ名による不一致が増えた**ためと分析されています [1]。
- **型の不整合（TS2322, TS2345）**: 代入時や関数呼び出し時の型不一致によるエラーです [3]。例えば「型Xを型Yに割り当てられない」「引数の型Xはパラメータの型Yに割り当てられない」等で、これらもTypeScriptでは頻出します。LambdaNetのような自動型付け出力では誤った型注釈により関数の戻り値型が期待と異なるなどで多数観測されました [3]。
- **未定義のシンボル/モジュール（TS2304, TS2307）**: 変数名やモジュールが見つからない場合のエラーです [4]。例えばモジュールが**存在しないか型定義ファイルが見つからない場合**に発生し（`TS7016`も「モジュールの宣言ファイルが見つからない」旨の関連エラー）、外部ライブラリの型定義漏れや動的ロードの型変換ミスが原因となります [4]。
- **関数の引数個数不一致（TS2554）**: 関数定義と異なる個数の引数を渡した場合のエラー [5]。JavaScriptでは可変長引数や省略が許容されますが、TypeScriptでは厳密に一致しないとエラーになります。
- **呼び出し不可エラー（TS2349）**: 関数でない値を呼び出そうとした際のエラーです [6]。例えば変数に関数型ではない型注釈を付けたのに実行時に関数として呼んでしまった場合などに起こります。

以上のように、TypeScriptのエラーコードはエラー種別ごとに典型的な原因を表しており、モデルやコード変換手法ごとにどのエラーが多いかで**型推論/変換の弱点**を分析できます [1] [2]。実際、後述するTypeWeaver論文では各モデル出力のエラー種類分布が詳細に分析され、エラー種別ごとの原因考察が行われています。

他の静的型付け言語でも、コンパイルエラーを体系化する試みがあります。JavaやC++では**シンボル未解決**、**型不一致**、**構文エラー**、**依存関係の問題**などが大きなカテゴリとして報告されています。例えばGoogle社内の大規模分析（26.6百万ビルド、ICSE 2014）では、**依存関係に起因するエラー**が最も一般的であり、開発者はそれらの解決に多くの時間を費やしていることが示されました [7]。具体的には、GoogleのC++/Javaビルドでは**他コンポーネントとの依存関係エラー**が頻発し、ビルド失敗の主要因となっていたと報告されています [7]。また、Zhangらの研究（ESEC/FSE 2019）では、GitHub上のJavaプロジェクトCIビルドログ約685万件を解析し、コンパイルエラーを**498種類**に分類しています [8]。その結果、上位20種のエラータイプで全エラーの95.4%を網羅でき、特に**「シンボルが見つからない」エラー（cant.resolve）**が全体の54.0%と過半を占め、次いで**「存在しないメンバアクセス」エラー（doesnt.exist）**が15.9%を占めることが分かりました [9]。上位3種類で全エラーの75.5%、上位10種類で90.2%を占めており、**ごく一部のエラー種別に集中している**ことが明らかです [10]。このような傾向は先行するGoogle社内研究 [11] とも共通しており、依存関係エラーやシンボル未定義エラーがどの環境でも主要なカテゴリとなることが確認されています [12]。総じて、**コンパイルエラーには繰り返し現れる代表的パターンが存在し、それらを分類体系化して捉えることで、エラー対策の優先度付け**が可能になります [13] [14]。

エラー収集・ログ解析と傾向の可視化

大規模なエラーログ収集と分析により、エラー発生の傾向を把握し対策を講じる研究も行われています。IDEやCI/CDパイプラインから得られるエラー情報を解析することで、**頻出エラーの可視化**や**問題箇所の特定**が可能になります。

前述のZhangらの研究 [15] は、Travis CI上のオープンソースJavaプロジェクトにおけるビルド失敗ログを網羅的に収集・解析したものです。この研究では、全ビルドの11%がコンパイルエラーで失敗しており、75%のプロジェクトで一度はコンパイルエラーによるビルド失敗が起きていました [16]。エラータイプ分布の可視化結果（Fig.3）から、先述の通り少数のエラー種が大半を占めることが示され、**エラー頻度のグラフ化**によってどのエラーがボトルネックかが一目で分かるようになっています [10]。さらにプロジェクトの属性（規模やスター数）、ブランチ戦略、テストコードか本番コードか、といった軸でエラー発生傾向を分析し、どの条件下でどのエラーが起きやすいかが議論されています [17] [17]。例えば、Googleの社内研究では**大規模単一コードベース**ゆえに一部のエラー種（未使用の型やraw型に関する警告など）の出現率が社外OSSとは異なる、といった比較もなされています [18] [19]。

こうした**ログ解析**により得られた知見から、具体的な対策提案も行われています。Zhangらは、頻発するエラーに対してIDEやCIツールが自動修復や防止機能を提供することで開発者支援につながると提言しています [20] [21]。例えば、全エラーの半数以上を占めるシンボル未解決エラーに対し、ビルド前の依存関係チェックや型定義の自動追加をCIで行う、といった方策が考えられます。また、プロジェクト間比較で特定のエラー（例：抽象メソッドの未実装エラー）がテストコードで多い場合には、テンプレートコード生成で未実装警告を減らす、など**エラー傾向の可視化**をもとにした改善が提案されました [17] [22]。

一方、IDEのテレメトリやクラッシュレポートを用いて**開発環境でのエラー収集**を行う試みもあります。Microsoft Visual StudioやVS Code等ではユーザの匿名エラーログを収集する仕組みがありますが、そのデータを研究に活用した公開事例は限定的です。ただしGitHub上のCIだけでなく、ローカル開発中にどんなコンパイルエラーが頻繁に出ているかを大量データで分析すれば、より開発者行動に即したエラー傾向が掴めるでしょう。Googleのケーススタディ [23] [7] では、自社ビルドシステム上で9ヶ月間に発生した何千万というエラーを分析し、**ビルド失敗の3割以上がコンパイルエラー**によること、その中でもC++で依存解決エラーが多発することを明らかにしています [7]。このような社内テレメトリの分析結果は、**ビルドシステムや言語ツールの改善ポイント**を示唆し、例えばエラーメッセージの改善やドキュメント整備（依存関係エラーの解決法提示など）に繋がったと考えられます。

自動修復ツールとエラーコード別のパターン

コンパイルエラーを自動で修正するツールや手法の研究も進んでおり、エラーの種類ごとに異なるアプローチを取ることで精度向上を図る例があります。大規模エラー解析から得られた知見として、**エラー種別ごとに単純な修正パターンが存在する**ことが確認されています [14]。Zhangらは上位10種のコンパイルエラーについて、325件の実際の修正コミットを手動分析し、それぞれのエラー種に対応する典型的な修正方法をまとめました [24]。例えば「シンボル未定義」のエラーでは「該当シンボルの宣言を追加する」か「誤綴を正す」パターンが多く、型不一致エラーでは「型キャストを追加」もしくは「変数宣言の型を変更」などが見られた、という具合です。この**エラー別修正パターン**の存在は、IDEのクイックフィクス機能等でも一部活用されています。実際、同研究によるとIDE（EclipseやIntelliJ）が提供する自動修正のサポートは限定的ながら存在し、エラー種によっては自動フィクスが可能であることが示唆されています [25]。

コンパイルエラー自動修復の専門ツールとしては、主に学生のコードを対象にしたものや、特定言語の文法ミス修正に特化したものがあります。例えばC言語の文法エラーを機械学習で修正するDeepFix（ICSE 2017）や、Pythonの構文エラー修復を行うDrRepair（ICLR 2020）などが知られています。こうしたツールでは、まずエラーの種類（構文エラー、型エラー、名前解決エラーなど）を分類し、それに応じた修正パターンを生成・評価します。DeepFixではコンパイルエラーのエラーメッセージを入力としてニューラルネットワークが修正コードを出力しましたが、例えば「セミコロン欠如（一般的な構文エラー）」の場合にはセミコロンを挿入する修正が高頻度で採用されるなど、モデルが**エラー内容に応じた修正を学習**していました。また最近では、コンパイルエラー修正を大規模言語モデル(LLM)と強化学習で解決する研究も登場しています [26] [27]。Sunら(2023)はC++コンパイルエラーの大規模データセットを生成し、LLMによる候補修正とコンパイラ検証を組み合わせたフレームワークを提案しています [28] [29]。このフレームワークではエラー種別ごとに多様なシナリオのコードを用意し、モデルが**特定のエラー種に対して適切な修正を学習できるよう設計**されています [30]。例えばテンプレート関連のエラーにはテンプレート文法を含むコード例で訓練する、といった工夫です。加えて、エラー種別単位でモデルの性能を評価し、どの種類のエラー修正が得意/不得意か分析することで、モデルの弱点を特定することにも役立てています [31]。

TypeScriptにおいても、自動型修復・補完の文脈でエラーコード別の対策を組み込む動きがあります。TypeScript特有のエラーに対し、**ケースごとの対処を行うツールチェーン**として注目されるのがYeeらのTypeWeaverです [32]。TypeWeaverは任意の型予測モデル（DeepTyperやLambdaNetなど）と組み合わせてJavaScriptプロジェクトをTypeScriptに一括移行するツールですが、移行時に生じるエラーを削減するためのいくつかの仕掛けを実装しています [32] [33]。具体的には: - **型依存の自動インポート**: 移行対象のプロジェクトが依存する外部パッケージに型定義が無い場合、自動的にその型情報（`.d.ts`ファイル）を取得・インポートします [33]。これにより、`TS2307`や`TS7016`（モジュールの型定義未解決）のエラー発生を事前に防ぎます。 - **モジュールシステムの変換**: Node.js向けのCommonJS形式の`require`を使ったコードを、ES Modules形式の`import/export`に変換します [34]。TypeScriptではESM形式でないと型チェックが完全には機能しないため、`TS2307`のようなモジュール関連エラーや型抜けを減らす効果があります。 - **型織り込み（weaving）**: モデルが予測した型を元のJSコードに挿入する処理を自動化します [35]。これ自体はエラー修復というより型付け工程ですが、切り離された予測結果を人手で挿入する手間を省き、正確に注釈をコードへ織り込むことで**一貫した型チェック**を実現します。 - **非型トークンの除去**: 予測モデルによっては型注釈として無効な文字列（構文的に誤った型名など）を出力する場合があります。そのような予測はTypeScriptパーサでエラーになるため、TypeWeaverではそれらを検出して除外・修正します [36]。この処理により、`TS1005`（区切り記号の欠如）等の**単純な構文エラー**を未然に防ぎます。

このようにTypeWeaverは**エラーコードに応じた前処理・後処理**を組み合わせることで、自動型注釈ツール全体の型適用成功率を高めています。実際、著者らの評価ではTypeWeaverを通すことで複数のモデル出力における**型チェック通過率が向上**し、特に依存関係の型解決(`TS7016`関連)やモジュール未解決エラー(`TS2307`)が大幅に減少したと述べられています [33] [32]。これらのアプローチは段階的な型修復デザインに通じるものであり、**エラー種別ごとにモジュール追加→構文修正→型矛盾修正→残りの細部調整**というフェーズ分割が有効であることを示唆しています。

型注釈導入によるエラー分布変化の評価

自動型付けや型注釈の導入は、元のコードに新たなコンパイルエラーを発生させる可能性があります。その**介入効果**を評価する研究として、TypeScriptコードへの自動型付け後にどのエラーがどれだけ増減したかを分析するものがあります。

YeeらのECOOP 2023論文では、前述のTypeWeaverを用いてJavaScriptパッケージ513個をTypeScript化し、3種類の型予測モデル（DeepTyper・LambdaNet・InCoder）の出力を比較しています [32] [37]。興味深いことに、モデルによって**エラー発生の傾向が大きく異なる**ことが報告されています。例えばLambdaNetはGraphベースで比較的具体的な型を当てる分、誤ったプロパティアクセスによる `TS2339` エラー（プロパティ不存在）が**24,123件**と突出しましたが、InCoder（大規模言語モデル）は汎用的な型や `any` を多用するため `TS2339` は**6,742件**と抑えられ、その代わり未定義変数エラー `TS2304` が多め（2,094件）に出る、といった差異です [38] [39]。DeepTyper（RNNベース）は `TS2339` が2,510件と比較的少なく、これは**汎用型で逃げがち**なためと考察されています [38]。このように自動型生成アプローチごとに**増えるエラー種が偏る**ことが定量的に示されており、型補完ツールの評価には**エラーコード別の分布変化**を検証することが重要だとわかります。

具体的な介入効果の例としては、**OpenTau**（Cassanoら,　　2023）というアプローチがあります [40]。OpenTauは大規模言語モデルによる型予測候補を探索し、TypeScriptコンパイラで型チェックして**エラーが少ない解を選択**するという手法です [41] [42]。OpenTauでは評価指標として「ファイルあたりの型エラー数」や「型検査に通ったファイル割合」を用いており、提案手法により**ファイルの47.4%がエラーなしコンパイル可能になった（従来比+14.5ポイント）**と報告しています [43] [44]。この改善の背景には、OpenTauが**複数候補の中からエラー数最小のものを選ぶ**ことで、例えば外部モジュール型未解決エラー（TS7016/TS2307）を避けたり、最小の型不整合に抑えたりする効果がある点が挙げられます [41] [45]。言い換えれば、OpenTauは**エラー分布を操作目標**として組み込んだ手法であり、型生成によって増えがちなエラーを抑制する方向にモデル出力を導いていると言えます。

一方で、エラー分布の変化から**型生成の質的特徴**を評価する動きもあります。先のTypeWeaver評価では、モデルごとのエラー内訳から「LambdaNetは詳細な型推測に強いが過剰に厳密でコードを壊しやすい（TS2339多発）」「InCoderは寛容な型付けでコード破壊は少ないが曖昧な型が増える（any乱用）」といった分析がなされています [1] [38]。このように、**型注釈導入によるエラー増減を分析することでツールの挙動を評価**し、改善すべきポイントを特定できます。例えば「あるツールAを使うと未定義モジュールエラーは減るがプロパティエラーが倍増する」のであれば、次の改良ではプロパティ推論アルゴリズムを工夫する、といった方向付けができます。実際、TypeWeaverやOpenTauの研究は、エラー種別の発生率変化を細かく測定し、**単純精度（正しい型予測率）では見えない観点からモデル評価**を行う重要性を強調しています [46]。特にTypeWeaver論文では「Accuracy指標だけではモデル優劣を評価できない」例として、LambdaNetが型一致精度は高いのに型チェック通過率で劣る逆転現象を示し、**コンパイルエラーの観点で評価する必要性**を示唆しました [47]。

教育的応用：エラー分類による学習支援

コンパイルエラーの分類と可視化は、プログラミング教育においても活用されています。初心者はコンパイルエラーに躓きやすく、エラーメッセージの難解さが学習継続の妨げになることが知られています [48] [49]。そのため、教育研究では**学生が頻繁に犯すエラーをデータ駆動で分類し、対策を講じる**試みがいくつか行われています。

一例として、ある研究では中学生向けにPythonプログラミングの**コーディングマニュアル**を作成するために、自動採点ツールで収集した**6,015件**の誤答コードを分析しました [50]。エラーを「言語仕様エラー（構文ミスなど）」や「ロジックエラー」などに分類し、それぞれについて頻出するパターンをマニュアルにまとめて学習者に提供したところ、明白な構文ミスに対してはエラー頻度が有意に減少する効果が見られました

[51]。これは、**共通のエラー原因を分類・明示して教える**ことで一部のエラーを未然に防げる可能性を示しています。一方で、この研究では学習者のプログラミング自信向上には繋がらず、マニュアルの限界も報告されています[52]。しかし重要なのは、**エラー分類に基づく指導**が実際に試行され、その有効性と課題が検証されている点です。

また別のアプローチとして、授業で**エラーログのヒートマップ**を用いる例も考えられます。例えば数十人規模のクラスでオンラインジャッジ等を利用している場合、どのエラーが何回出たかを集計し、講義中に「今回多かったエラーTOP3」をフィードバックすることで、学生が自分だけでなく多数が陥りがちなポイントを認識できるようにします。これは直接の研究事例というより教育実践のアイデアですが、近年の学習分析(learning analytics)の文脈では**コンパイルエラーの統計情報を講義設計に活かす**ことが注目されています[53]。実際、Jadudが提案したError Quotient (EQ)[54]のように、コンパイルエラー反復率を定量化して学生のつまづき度合いを測る指標もあります[55]。EQは主に構文エラー連発の程度を測るものですが、これを複数のエラータイプに拡張し、どのカテゴリーのエラーで特に躓いているかを可視化する試みも報告されています[56]。例えば「データ型に関するエラーEQ」と「文法ミスに関するエラーEQ」を別々に算出し、前者が高い学生には型の概念補強を行う、といった指導へのフィードバックが可能です[57][58]。

さらに、エラー分類は**エラーメッセージ自体の改善**にも応用されています。研究者や教育者は長年、初心者にとって分かりやすいエラーメッセージの設計を模索してきました[48]。例えばNienaltowskiら(2008)やDennyら(2014)は、コンパイラが出す生のメッセージを補足・言い換えることで理解を助けるシステムを開発しました[59]。これらのシステムでは、内部的にエラーを型、構文、論理などに分類し、それぞれに応じた**平易な説明文やヒント**を表示します。近年ではPratherら(2017)による**Enhanced Compiler Error Messages**の研究もあり、Javaのコンパイルエラーに追加情報（原因の背景や修正例）を付与する実験が行われています[60]。結果、強化メッセージを見た学生はエラー修正に要する時間が減少し、コンパイル成功率も向上したとの報告があります[61][62]。このように**エラー分類を踏まえたメッセージ改善**は学習効率に寄与すると期待され、TypeScriptにおいても例えば「プロパティ不存在エラー(TS2339)」であれば「オブジェクトの型定義にプロパティが足りない可能性」を提示する、といったインテリセンス的な支援が考えられます。

大規模エラーデータセットと実証研究

近年、GitHub等から大規模に収集したエラーデータセットを構築し研究に供する動きも盛んです。前述のManyTypes4TypeScriptなどはその一例で、TypeScriptの型推論評価用に**大量のJavaScriptコードと対応するエラー情報**を集積したデータセットです[63]。ManyTypes4TSでは、型予測モデルの学習・評価のために、まず**TypeScriptコンパイラで型エラーを検知**し、完全に型チェックが通るコードや一部エラーのあるコードなど様々なケースを含むよう工夫されています[64]。特に、OpenTauの評価にはこのデータセットが使われており、入力プログラムが**文法的に正しく（構文エラーがなく）**、かつ極端なケース（例えば全てanyで型注釈すれば良いような trivially typable なケース）を除外するフィルタが施されています[65]。こうした前処理により、**プログラム中の型エラー数やtypednessスコア**といった指標が計測され、モデル間比較に供されています[65]。

また、コンパイルエラー修正用のデータセットとしては先述のCCrepairBenchがあります。これは**C++のコンパイルエラーに特化した初の大規模コーパス**であり、Clang++のエラーメッセージ種別ごとに多様なコード例を生成・収集したものです[28][29]。CCrepairBenchでは数百種類に及ぶエラー型（例えばテンプレートエラー、型変換エラー、宣言ミス等）について、対応する誤ったコードと正しい修正例のペアが含まれており、研究コミュニティに公開されています[28][27]。このようなデータセットは、コンパイルエラーの原因分析や修復アルゴリズムの性能評価にとって極めて有用です。特に、従来は各研究者が個別に収集していたエラーケースを共有資源として提供することで、**再現可能な評価**や**手法間の公平な比較**が可能になります。

GitHub ActionsやTravis CIのログをオープンデータ化する取り組みもあります。例えばGHALogs[66]はGitHub Actionsの数百万件にのぼるビルドログデータセットで、CIパイプライン研究に利用できるよう公開されています。ここにはテスト失敗や依存解決エラーなど様々な失敗要因が含まれますが、工夫次第で**コン**

パイルエラー部分のみフィルタして分析に使うこともできます。Travis　CI関連ではZhangらのデータ（FSE 2019で公開）[67]があり、約3,799プロジェクト・685万件のCIビルド失敗ログが整理されています。このデータはJavaコンパイラ`javac`のエラーメッセージと発生頻度を含んでおり、研究者がさらなる分析やツール評価に用いることができます[68]。

またMicrosoftはVisual Studioの**顧客改善プログラム**で収集したエラーデータを内部分析していると考えられますが、公的なデータセット化は確認されていません。しかし、今後IDEベンダーがコンパイルエラー統計を公開すれば、例えば「全世界で最も多いC#のコンパイルエラーTop10」といった興味深い指標が得られるでしょう。

エラー分類指標を用いた品質メトリクス

コンパイルエラーに関する分類指標は、ソフトウェアの品質やツールの性能評価にも活用できます。例えば、前述の型注釈自動導入ツールを評価する際、単に「正解の型を当てた率（Accuracy）」を見るだけでなく、**導入後に生じたエラーの種類と件数**を指標化することで、ツールの実用性を測ることができます[69]。具体例として、Yeeらは**型チェック通過率（エラーゼロのファイル割合）**と**型エラー数平均**を主要指標とし、さらに**非自明な型注釈率**を加えた複合指標でモデル比較を行いました[43][70]。非自明な型注釈率とは、`any`や`Function`のようなトリビアルな型に逃げていない度合いを示すもので、単にエラーが出ないだけでなく有用な型情報を提供できているかを評価するためのものです[71][72]。この指標設計により、**「エラーを完全になくすにはany乱用が手っ取り早いがそれでは品質が低い」**というジレンマに対処しつつ、エラー削減と有用性のバランスを定量的に測れるようにしています[71][72]。

質問にある「TS2339発生率を“破壊率”と定義する」といった発想も、興味深い品質メトリクスの一つと言えます。これは型注釈導入によって**既存の動的コードを壊してしまった度合い**を測る指標と解釈できます。TS2339（プロパティ不存在）は本来JavaScriptでは許容されていたプロパティアクセスが型制約でエラーになるケースを指すため、このエラーの頻度が高いほど「導入した型定義が厳しすぎてコード互換性を壊している」ことを意味します。したがって**破壊率（TS2339発生率）**が低いツールほど、既存コードの挙動を阻害せずに型付けできている＝移行の安全性が高い、と評価できるでしょう。実際、TypeWeaver評価ではLambdaNetがこの「破壊率」が極めて高く、一方DeepTyperやInCoderは低かったため、後者の方が**開発者が扱いやすい出力**をしていると解釈できます[38]。もっとも、破壊率が低すぎる場合はany型だらけで型チェックの意味が薄い可能性もあるため、先述の**有用性指標**と併せて見る必要があります[71]。

コンパイルエラー分類を組み合わせた品質メトリクスは他にも考案されています。例えば、前述のError Quotient(EQ)は教育分野のメトリクスでしたが、これをプロジェクト全体に適用し「ビルド当たり平均エラー数」を継続的に追うことで、コード品質向上を測る試みも可能です。ビルドあたりエラー数が減少傾向にあればリファクタリングやCIチェック導入の効果が現れていると判断できますし、特定のエラー種（例：null参照関連のエラー）が比率として減っていれば、コードベースにおけるその種の欠陥低減を示唆できます。

総じて、コンパイルエラーの分類・分析に関する研究は**エラーの傾向を定量化し、焦点を絞った対策や評価指標を提供**してきました。TypeScriptの型エラーに関しても、エラーコードを軸に段階的に問題を解決していくアプローチは、こうした先行知見と整合的です。エラーの分類体系に基づき優先度の高い問題から順に補完・修復することで、効率的かつ破壊の少ない型移行を実現できるでしょう。本調査で示したように、各種研究成果が**フェーズ分割型の型補完/修復手法**の有効性を裏付けており、その設計妥当性は十分に支えられていると言えます。

参考文献: コンパイルエラー分類・分析・修復に関する各種研究論文[1][10][7][33][38][14]ほか。

---

1 2 3 4 5 6 46 47 TypeWeaver_ Learning Type Annotations from Examples 論文調査報告.pdf
file://file-6R3JyfdjD5GzHmPvMquMmZ

7 23 research.google.com
https://research.google.com/pubs/archive/42184.pdf

8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 24 25 67 68 A Large-Scale Empirical Study of Compiler Errors in Continuous Integration
https://chenbihuan.github.io/paper/fse19-zhang-compiler-error.pdf

26 27 28 29 30 31 CCrepairBench: A High-Fidelity Benchmark and Reinforcement Learning Framework for C++ Compilation Repair
https://arxiv.org/html/2509.15690v1

32 33 34 35 36 37 38 39 69 Do Machine Learning Models Produce TypeScript Types That Type Check?
https://drops.dagstuhl.de/storage/00lipics/lipics-vol263-ecoop2023/LIPIcs.ECOOP.2023.37/LIPIcs.ECOOP.2023.37.pdf

40 43 44 [2305.17145] Type Prediction With Program Decomposition and Fill-in-the-Type Training
https://arxiv.org/abs/2305.17145

41 42 45 65 70 71 72 [2305.17145] Type Prediction With Program Decomposition and Fill-in-the-Type Training
https://ar5iv.org/pdf/2305.17145

48 49 50 51 52 53 59 A Taxonomy of Compiler Error Messages. | Download Scientific Diagram
https://www.researchgate.net/figure/A-Taxonomy-of-Compiler-Error-Messages_fig1_323328566

54 An exploration of error quotient in multiple contexts
https://dl.acm.org/doi/10.1145/2828959.2828966

55 [PDF] A New Metric to Quantify Repeated Compiler Errors for Novice ...
https://www.brettbecker.com/wp-content/uploads/2016/07/ITiCSE-Becker.pdf

56 58 Comparison of Three Programming Error Measures for Explaining ...
https://arxiv.org/html/2404.05988v1

57 A New Metric to Quantify Repeated Compiler Errors for Novice ...
https://researchrepository.ucd.ie/entities/publication/4a56127f-90db-4d9f-8b9e-9c6aa0d6d400

60 On Designing Programming Error Messages for Novices
https://dl.acm.org/doi/fullHtml/10.1145/3411764.3445696

61 [PDF] Effective Compiler Error Message Enhancement for Novice ...
https://www.brettbecker.com/wp-content/uploads/2018/03/becker2016effective.pdf

62 Effective compiler error message enhancement for novice ... - LinkedIn
https://www.linkedin.com/pulse/effective-compiler-error-message-enhancement-novice-students-becker

63 64 ManyTypes4TypeScript: a comprehensive TypeScript dataset for ...
https://www.researchgate.net/publication/364460595_ManyTypes4TypeScript_a_comprehensive_TypeScript_dataset_for_sequence-based_type_inference

66 [PDF] GHALogs: Large-Scale Dataset of GitHub Actions Runs - s3@eurecom
https://s3.eurecom.fr/docs/msr25_moriconi.pdf