自己の多様な作業履歴を活用するプログラミング支援環境のデザインと試作

渡部 丈　山本 恭裕　中小路 久美代

プログラミングでは，コーディングに加え，実装方針等のメモ，Web 検索，作業中の思考に伴う発話，生成 AI との対話といった様々な作業が行われる．こういった知的活動の履歴は，それぞれのツールや環境に分散したり，あるいはそもそも残されていなかったりする．我々はこれまで，こうした多様な活動の履歴を通して過去の経験を想起し，現在のプログラミングに活用することを「過去の自分との対話」と捉え，HALO (History-based Augmenting Learning Opportunities) というプログラミング学習環境を提案してきた．本稿では，このアプローチを学習支援にとどまらず，広くプログラミング作業一般を支えるものとして再構成する．これらの作業をそれぞれ支援・記録し，生じた履歴を同一の時間軸上で横断的に閲覧するプログラミング支援環境のデザインと試作を報告する．試作した環境は，ユーザの思考を外在化する Monologue Tools，ユーザの入力に対しシステムの出力を受けて進める Dialogue Tools，および両者の履歴を Web 検索履歴とともにタイムライン上に表示する履歴閲覧ブラウザから構成される．Dialogue Tools では，コードと実行結果のペアや，プロンプトと生成 AI の応答といったやりとりを，それぞれ二次元平面上に配置しながら作業を進める形のユーザインタフェースを提供し，その配置自体を履歴として残す．空間上の自由な配置は，作業中には考えを展開するための材料として働き，後からは当時の状況を想起するための手がかりとしても機能することを意図したものである．この環境を用いたシナリオを通して，想定する利用のされ方を示して本稿を結ぶ．

1　はじめに

プログラミングは，あらかじめ定められた手順に従って進める定型的な作業ではなく，問題の定義と解の構築が相互に影響しながら展開する設計的な活動である．コーディングを進める中で問題の枠組み自体が精緻化され，試行錯誤や発想生成といった多様な認知的活動が並行して行われる．このような特徴から，プログラミングは創造性を要する知識活動として捉えることができる．

こうした活動は，コードを記述する作業だけで進むわけではない．実装方針やそのとき気づいたことは文章として書き留められ，分からないことは Web で検索され，「ここがおかしいかもしれない」といった判断はしばしば独り言として発せられる．近年ではこれらに加えて，生成 AI との対話を通して考えを具体化することも一般的になりつつある．これらの作業はそれぞれ痕跡を残しうるが，コードはエディタに，検索はブラウザの履歴に，メモは別のアプリケーションにというように，痕跡の残る場所は互いに分散しており，発話のようにそもそも残らないものもある．その結果，ある時点で何を考え，何を試し，どこまで分かっていたのかという作業の流れは，個々の道具の中には残らない．

一方で，これらの作業履歴は，過去に作成した情報を後から検索・確認・再利用するための資源となるだけではない．短いコメントやファイル名を見たことをきっかけに，当時なぜそのコードを書いたのか，どのような問題に困っていたのか，別の案をなぜ採用しなかったのかが思い出されることがある．我々はこの性質を，パースの記号論における指標（index）の概念を手がかりに位置づけている [10] [12]．エラーログは

Leveraging One's Own Diverse Activity History in a Programming Support Environment

Jou Watabe　Yasuhiro Yamamoto　Kumiyo Nakakoji, 公立はこだて未来大学システム情報科学研究科　公立はこだて未来大学システム情報科学部, Dept. of Systems Information Science, Future University Hakodate.

エラーが発生した活動の痕跡であり，メモはその時点での思考を書き留めた記録であり，検索履歴は特定の情報を必要としていた行為の痕跡である．これらは過去の活動との事実的な結びつきによって当時の作業や思考を指し示すが，そこから何が想起されるかは一義的に定まらず，現在取り組んでいる課題や関心によって変化する．すなわち履歴の利用は，保存された情報をそのまま取得することにとどまらず，履歴を手がかりとして過去の状況を現在において再構成することを含む．我々は，このように多様な活動の履歴を通して過去の経験や知識を思い出し，現在のプログラミングに活用することを「過去の自分との対話」と捉えている [10]．

このような想起の手がかりは，言語的・記号的な記述として残された情報だけに限られない．我々はこれまで，デザインの初期段階において，考えていることや作りかけの部品をオブジェクトとして二次元空間に配置することが，言語化や形式化を強いることなく思考を外在化する hands-on representation として働くことを論じてきた [7]．オブジェクトの大きさや色といった単項的な性質，近い・離れている・重なっているといった他のオブジェクトとの関係，空間全体の中でのおおまかな位置といった視覚的な性質は，作業者がまだ言葉にできていない判断や確信の度合いをそのまま表しうる．そして，そのようにして生じた配置は，作業を進めながら作業者自身へ視覚的に働きかけると同時に，後から見返したときには，その表現が生み出された思考の瞬間を指し示すインデックス的なインタフェースとしても機能する [4]．すなわち二次元空間上の配置は，作業中には考えを展開するための材料として，後からは当時の状況を想起するための指標として，二重に働きうる．本研究は，この二重性を，プログラミングに伴う履歴を扱ううえでの中心的な手がかりとして位置づける．

なお履歴が果たす役割は，現在からの時間的距離によっても異なる [10]．直前に書いたコードやメモは進行中の思考を保持するために，数日前から一週間程度の履歴は継続中の課題を再開し，それまでの判断や試行錯誤を振り返るために用いられる．これに対して数週間から数年前の遠い過去の履歴は，現在の課題と直接には関連しないことが多く，検索語をあらかじめ定めて目的の情報を取り出すことも難しい．しかし，探すべき対象がまだ言語化されていない段階であっても，個人的な経験の連鎖を介して予期しない候補へ到達しうるという点に，外部情報の検索とは異なる自己の履歴の意義がある．

以上の立場からは，履歴を扱う環境に対して，必要な情報を漏れなく保存し検索要求に一意の正解を返すこととは異なる設計上の要請が導かれる [10]．第一に，履歴が生成された時間的・活動的な文脈をたどれることである．第二に，現在の作業を中断しすぎない形で履歴と再遭遇できることである．第三に，直接的には関連しない履歴へ探索を広げられることである．そして第四に，記号的な記述に限らない形で考えを外在化でき，かつその外在化された表現自体が履歴として残ることである．本研究ではこれらを踏まえ，コードの記述以外の作業についてもそれぞれ支援したうえで，後から振り返るのに意味のある単位で履歴を残し，種類の異なる履歴を同一の時間軸上に置くという方針をとる．具体的には，ユーザの操作に対してシステム側が結果を返し，それを受けながら作業を進めるツールを Dialogue Tools，テキストメモや発話のようにユーザの思考を外在化するツールを Monologue Tools と位置づけてツール群を構築する．このうちコードの記述・実行と生成 AI との対話については，コードや対話を一つの流れとして上から下へ並べるのではなく，二次元平面上に配置しながら作業を進める形をとり，配置や大きさを含めて履歴として保存する．履歴は画面を定期的にキャプチャして得るのではなく，コードセルの実行や生成 AI へのプロンプトの送信といった，各ツールにおいて意味を持つ操作を契機として生成する．そのうえで，これらの履歴と Web ブラウザ上の検索・閲覧履歴を同一の時間軸上に並べ，履歴の種類ごとにレーンを分けて表示する履歴閲覧ブラウザを構築する．

これまで我々は，プログラミングに伴う活動の履歴を時系列に統合して閲覧することでプログラミング学習支援に活用する HALO (History-based Augmenting Learning Opportunities) 環境 [11] を提案してきた．本稿では，その対象を学習支援に限らずプログラ

ミング作業一般へと広げた，プログラミング支援環境のデザインと試作を報告する．履歴が生じる側の作業そのものを支援するツール群として，HALO を構成していたエディタ群を再設計するとともに，生成 AI との対話を扱うツールを新たに加えた．

以下，2 章で関連研究を述べ，3 章で各ツールのデザインを，4 章で実装を説明する．5 章では加速度センサデータの可視化を題材とした利用シナリオを示し，6 章で本稿を結ぶ．

2 関連研究

プログラミング環境における履歴活用の歴史は長く，版管理や編集時の Undo/Redo の取扱，ノートブックの実行履歴といった数多くのアプローチが報告されている．本節では，作業履歴を用いた支援に関する研究，および生成 AI を用いたインタラクティブなプログラミング支援に関する研究について概観する．

2.1 作業履歴を用いた支援

計算機上で行われる作業について，その過程で生じる履歴を記録し，後から参照したり可視化したりすることで作業を支援しようとする研究は，これまで様々に行われている．Hill らは，文書のどの部分がどの程度編集されたり閲覧されたりしたのかを，物理的な物体が使用によって摩耗していく様子になぞらえて可視化する Edit Wear and Read Wear を提案している [3]．Sáenz らの Dear Diary は，ソースコードに実装時のメモを関連付けて残すことで，そのコードを記述した際に何を考え，どのような判断を行ったのかを後から振り返ることを支援する [5]．

また，過去の自分自身との関係に着目した研究として，Cockburn は，現在の自分と過去の自分とを時間的に隔てられた協調作業の主体として捉える Reflexive CSCW を論じている [2]．本研究も，作業の結果だけではなく，そこに至るまでに生じる履歴をプログラミング支援に利用する立場をとる．ただし本研究では，コードだけを履歴として扱うのではなく，プログラミング中に記述されたメモ，作業中の発話，Web 検索や Web ページの閲覧，生成 AI との対話といった異なる種類の作業を対象とし，それぞれを支援するとともに，そこで生じた履歴を時系列に沿って関連付けながら扱う．

2.2 生成 AI を用いたインタラクティブなプログラミング支援

生成 AI を用いたプログラミング支援については，自然言語による要求からコードを生成するものにとどまらず，プログラミングの様々な段階に関わる方法が検討されている．Yen らは，コードや実行結果の上にユーザが描いた自由形式のスケッチを生成 AI が解釈し，その内容をコード編集へ反映する Code Shaping を提案している [8]．生成された結果が意図と異なる場合にはスケッチを修正して再び解釈させることで，一度の指示と一度の生成という関係ではなく，意図の外在化と結果の確認を繰り返す反復的なインタラクションとしてコード編集を扱っている．

Zamfirescu-Pereira らは，一つの問題に対して単一の解決方法を提示するのではなく，問題の異なる捉え方とそれぞれに対応する解決案を提示することで，プログラムの設計空間を探索する過程を支援する IDE を提案している [9]．人間あるいは生成 AI が行った設計上の判断を明示することで，最終的に生成されたコードだけでなく，そこに至るまでにどのような判断が行われたのかをユーザが把握できるようにしている．

Chen らは，ユーザからの依頼に応答する従来のチャット型の利用方法とは異なり，作業状況に応じて生成 AI 側から支援を提示するプログラミング支援を検討し，提示する内容だけでなく，提示のタイミングや情報量，提案された変更のコードへの反映方法といったインタラクション上の問題を扱っている [1]．これらの研究では，生成されるコードそのものだけではなく，ユーザがどのように意図を表現し，生成結果をどのように受け取り，そこからどのように次の作業へ進むのかという，人間と生成 AI との継続的なやり取りが対象となっている．一方，そこで生じたやり取りが履歴として残り，後から他の作業の履歴とともに参照されうることは，これまで十分に扱われていない．

本研究では，このような生成 AI によるコード生成を独立した支援対象として扱うのではなく，コードの

記述・実行，実装方針等についてのメモ，作業中の発話，Web 検索，生成 AI との対話といった作業が相互に関係しながら進められることに着目し，個々の作業を支援するとともに，そこで生じる履歴を後から関連付けながら閲覧できるプログラミング支援環境を構築する．

3 プログラミング支援環境のデザイン

3.1 提案システムのデザイン方針

本研究では，プログラミングを，あらかじめ定められた完成像に向けてコードを記述していく作業としてではなく，コードを書き，結果を確認し，考え直すことを繰り返しながら，作るもの自体を徐々に具体化していく活動として捉える．その過程では，コードの記述・実行だけでなく，実装方針について考えること，気づいたことを書き留めること，分からないことを調べること，生成 AI との対話を通して考えを具体化することなど，様々な作業が行われる．本システムは，このようなプログラミングの過程を支援するとともに，そこで生じる多様な作業を履歴として残し，横断的に閲覧可能にすることを目指す．

本論では，プログラミング中に行われる作業を，Monologue 的な作業と Dialogue 的な作業として捉える．Monologue 的な作業とは，その時点で考えていることや気づいたこと，迷っていることなどを外在化する作業であり，テキストメモと作業中の発話がこれにあたる．一方，Dialogue 的な作業とは，ユーザが何らかの形で意図を表し，それに対して返された結果を受けながら，次に何をするか，何を作っていくかを考えていく作業である．例えば，コードを記述して実行すると実行結果やエラーが返され，ユーザはその結果を見てコードを変更したり，実装方針を見直したりする．生成 AI との対話においても，自分が作りたいものや分からないことをプロンプトとして表現し，得られた応答を受けて，さらにプロンプトやコードを変更していく．本システムでは，前者を支援するツール群を Monologue Tools，後者を Dialogue Tools と位置づけ，これらの履歴と Web ブラウザ上の検索・閲覧履歴を同一の時間軸上で閲覧する履歴閲覧ブラウザを設ける（表 1）．

各ツールはそれぞれ独立した画面を持つが，画面上部にはツール間を移動するための共通のナビゲーション領域を設けている．メモツール，コードツール，AI 対話ツールは画面を切り替えて使用する．一方，発話ツールと履歴閲覧ブラウザは，他の作業を行いながら利用できるよう別ウィンドウで表示する．以下，各ツールについて述べる．

3.2 メモツール

プログラミング中には，コードを記述するだけでなく，これから実装する内容や実装方針，発生している不具合，試してみたいことなどを文章として書き留めることがある．メモツールでは，このような文章を通常のテキストエディタと同様に入力・編集しながら，その途中状態を履歴として残せるようにしている（図 1）．画面の中心にある編集領域で文章を継続的に編集し，状態を残しておきたい時点でスナップショットを保存する．スナップショットには，その時点で編集領域に記述されている文章全体と保存時刻が記録され，画面左側に一覧として表示されるため，ユーザは現在の文章を編集しながら，それまでに保存した文章の状態を参照できる．一覧から選択したスナップショットは文章全体が現在の編集領域へ読み込まれ，その状態から編集を続けることができる．このとき，復元によって現在編集中の文章が失われることを避けるため，復元を行う直前の文章も自動的に退避する．したがって，過去の状態へ戻った後でも，必要に応じて復元前の文章へ戻ることができる．

文章の編集とスナップショットの保存が行われた時点は，編集領域の横に時系列として表示する．文章の編集は小さな点，スナップショットの保存はそれより大きな点として表示し，各点にマウスポインタを重ねると，文章の追加や削除，変更といった操作内容を確認できる．この表示は過去の文章を直接復元するためのものではなく，文章がどのような時間間隔で編集されていたのかを確認するために使用する．

さらに，現在編集中の文章を手がかりとして過去の文章を探すこともできる．ユーザが編集領域内の文字列を選択すると，その文字列を含む過去のスナップショットを検索し，該当する箇所を強調したうえで時

表 1　本システムを構成するツールとそこで生成される履歴

ツール（本稿での呼称）	履歴の生成契機	履歴の単位	オブジェクトストア
Monologue Tools			
メモツール	スナップショットの保存	保存時点の文章全体	`textSnapshots`
発話ツール	音声認識結果の確定	確定した発話一件	`speeches`
Dialogue Tools			
コードツール	コードセルの実行	ワークスペース全体	`stickies`
AI 対話ツール	プロンプトの送信	プロンプトと応答の組	`chats`
（別実装）			
Chrome 拡張機能	CSV のインポート	検索・閲覧イベント一件	`searchHistory`
履歴閲覧ブラウザ	—	—	—

図 1　メモツール

系列に沿って表示するため，同じ表現が過去の文章のどこに現れていたかを確認できる．検索されたスナップショットの一つを選択すると，そのスナップショットだけでなく，その前後に保存されていた文章も横方向に並べて表示する．これにより，現在の文章中にある特定の表現を起点として，その表現が記述された頃にどのような文章を書いていたのか，その前後で文章がどのように変化していたのかをたどることができる．検索結果から参照したスナップショットは別の領域にも蓄積されるため，複数の過去状態を参照しながら現在の文章を編集することもできる．

3.3　発話ツール

プログラミングを行っているときには,「ここがおかしいかもしれない」「先にこの処理を直そう」といった，その場で考えたことを独り言として発することがある．発話ツールは，こうした発話を，コードや文章とは異なる形で外在化された作業中の情報として記録する（図 2）．発話の収集を開始すると，コンピュータのマイクから入力された音声を認識し，認識内容が確定した発話について，認識されたテキストと時刻を一件の履歴として保存して，画面上の発話一覧へ順次追加する．発話の途中に間がある場合などには，一続きの独り言が複数の発話履歴として保存されることもある．保存された発話は古いものから新しいものへ時系列に沿って表示し，収集中に新しい発話が確定すると一覧の末尾に追加されるため，ユーザは作業を続けながら直近の発話を確認できる．本ツールはコード編集等の画面とは別のウィンドウで動作するため，コードを記述したり生成 AI と対話したりしている間にも収集を続けることができる．

過去の発話を振り返る際には，キーワードと日時の二つの方法で履歴を検索できる．キーワードによる検索では，入力された文字列を含む発話を抽出し，検索文字列の変更に応じて結果が更新されるため，入力しながら対象を絞り込むことができる．日時による検索では，ユーザが指定した日時を始点として，そこから 24 時間以内に記録された発話を対象とする．検索結果が複数ある場合には，スライダを動かすことで順番に選択でき，選択した発話は一覧上で強調されるとともに，発話テキストが音声として読み上げられる．文字として一覧を読むだけでなく，過去に自分が発した内容を音声として聞き直すことで，作業時の状況を振り返ることを想定している．また画面下部には，前日に記録された発話に含まれる語を，出現回数を集計して頻繁に現れた語ほど大きく表示する領域を設け，前日の作業中にどのような言葉を多く発していたのかを大まかに眺められるようにしている．

図 2　発話ツール

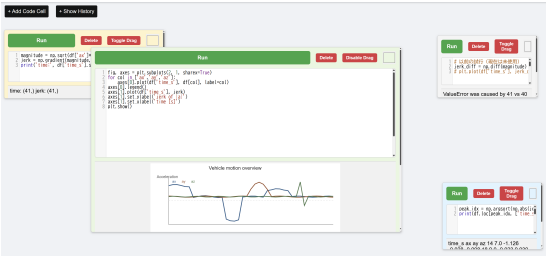

図 3　コードツール

3.4　コードツール

コードツールでは，コードを一つの長いファイルとしてのみ扱うのではなく，複数のコードセルに分けて二次元平面上へ配置しながら作業を進める（図 3）．各コードセルは，Python コードを記述する領域，実行結果を表示する領域，メモを記述する領域から成る．それぞれのセルはドラッグして移動でき，大きさや背景色も変更できるため，関連するコードを近くに配置する，異なる実装案を離して配置する，現在注目しているセルを大きくするといった形で，コードを二次元的に整理しながら作業できる．これらの配置は，どのコードをどう位置づけているかを言語化することなく外在化する手段であるとともに [7]，後から履歴として見返した際に，当時どのように考えていたかを指し示す手がかりともなる．複数のセルを重ねて配置することもできる．

コードを実行すると，そのコードから出力された文字列や生成された図が同じセル内に表示される．実行時にエラーが生じた場合には，エラーの内容も実行結果として表示する．コードとその結果が同一のセル内に保持されるため，どのコードからどのような結果が得られたのかを対応付けた状態で扱える．メモ領域には，コードが何をするものなのかという説明だけでなく，実装方針，不具合，試した結果，次に行う予定の作業などを記述できる．これにより，コードだけでは残りにくい作業時の意図や判断も，コードと対応付けて記録する．

任意のコードセルを実行すると，実行したセルだけでなく，その時点でワークスペース上に存在しているすべてのコードセルについて，コード，メモ，実行結果，位置，背景色などを保存する．したがって，履歴は一つのコードセルだけを表すものではなく，「そのコードを実行した時点におけるワークスペース全体の状態」を表すものとなる．保存された履歴は，現在のワークスペース上に重ねて表示する履歴表示領域から閲覧でき，履歴の時刻を変更すると，その時点で保存されたコードセル群が，保存時の位置関係を保った状態で表示される．これにより，過去にどのコードを記述していたかだけでなく，当時どのようなコード群を組み合わせ，どのように配置して作業していたかも振り返ることができる．ここで履歴として残るのは個々のコードの内容だけではなく，当時それらをどのように位置づけていたかという二次元平面上の配置そのものであり，これは後から当時の考えを想起する手がかりとなることを意図している．

3.5　AI 対話ツール

AI 対話ツールでは，一般的なチャットインタフェースのようにプロンプトと応答を上から下へ連続して並べるのではなく，一回のプロンプトとそれに対して得られた応答を一つのオブジェクトとして扱い，複数のオブジェクトを二次元平面上に配置して，それぞれを移動したり大きさを変更したりしながら対話を進める（図 4）．これにより，どのプロンプトが試行の枝分かれであり，どれが本筋であったかといった判断を，明示的に記述することなく配置として残すことができる．ユーザが新しいオブジェクトを作成してプロンプトを入力し，生成 AI へ送信して応答が得られると，プロンプトと応答が同じオブジェクト内に表示される．プロンプトの入力内容は送信時だけでなく入力途中にも保存されるため，文章を書きかけた状態で別の

作業へ移動した場合にも，後から続きを編集できる．

一つの会話の中で複数のプロンプトを送信した場合には，正常に応答が得られたプロンプトと応答を対話履歴として保持し，その後のプロンプトを送信する際には，それまでの対話履歴も生成 AI へ与える．したがって，画面上では個々のオブジェクトを任意の位置に配置できる一方で，生成 AI との対話そのものにはプロンプトを送信した順序が保持され，オブジェクトを後から並べ替えても会話の順番は変化しない．また，同じオブジェクトのプロンプトを書き換えて再度送信した場合も，新しいプロンプトと応答の組として対話履歴へ追加する．作業内容の異なる対話を分けるため，複数の会話を作成することもできる．会話を切り替えると，二次元平面上に表示するオブジェクト群と，生成 AI へ与えるそれまでの対話履歴も切り替わる．

履歴としては，生成 AI へ実際に送信したプロンプトと応答だけでなく，プロンプトオブジェクトそのものの変化も保持する．プロンプトを作成した後に文章を追加したり削除したりした場合，その変更をオブジェクトの履歴として残し，現在の画面から削除したオブジェクトについても履歴を保持する．さらに，生成 AI から応答を取得した時点で，そのとき二次元平面上に存在していたオブジェクト群の位置，大きさ，プロンプト，応答等を配置履歴として保存する．過去の配置履歴を閲覧する際には，異なる時点のオブジェクト群を重ねて表示し，特定の時点を選択して前面に表示できる．これにより，個々のプロンプトの内容だけでなく，生成 AI とのやり取りを進める中で，プロンプト群をどのように作り，配置し，変更していったかという作業の経過も振り返ることができる．

3.6 履歴閲覧ブラウザ

履歴閲覧ブラウザは，これまでに述べた各ツールで記録された履歴に加えて，Web ブラウザ上で行われた検索や Web ページの閲覧履歴を，同一の画面から時系列に沿って閲覧するためのツールである（図 5）．対象となる履歴は，コードとその実行結果，文章のスナップショット，発話，生成 AI へ送信したプロンプトとその応答，検索結果ページおよび閲覧した Web

図 4　AI 対話ツール

ページである．

画面上部では，閲覧する日付，履歴の種類，検索文字列を指定できる．履歴の種類については，コード，文章，Web 検索・閲覧，発話，生成 AI との対話を個別に表示または非表示にできる．また，検索文字列を入力すると，各履歴に含まれるコード，文章，発話，プロンプト，応答，検索キーワード，Web ページの情報などを対象として履歴を絞り込み，複数の条件を指定した場合には，それらの条件を満たす履歴だけを表示する．画面中央には，対象となる履歴を時系列に並べる領域を設けている．各履歴は，コードであればコードの一部，文章であればスナップショットの内容，発話であれば認識された発話，生成 AI との対話であればプロンプトと応答といったように，履歴の種類に応じた内容をプレビューするカードとして表示する．ユーザが履歴を一つ選択すると画面右側にその詳細を表示するため，多数の履歴を概観しながら，気になったものについて内容を詳しく確認できる．

画面下部には，履歴が一日のどの時点で発生したかを示すタイムラインを表示する．一日全体を示すタイムラインには，各履歴を保存時刻に対応する位置へ点として配置し，ユーザはこのタイムライン上で時間範囲を選択して，閲覧対象とする時間帯を変更できる．履歴が密集している時間帯を狭い範囲で見ることも，一日の広い範囲をまとめて見ることもできる．選択した時間帯については，履歴の種類ごとにレーンを分けたタイムラインも表示する．コード，文章，Web 検索・閲覧，発話，生成 AI との対話がそれぞれ異なるレーンに表示されるため，例えば，コードを実行する直前にどのような Web 検索を行っていたのか，生成 AI との対話の前後にどのようなメモや発話が残さ

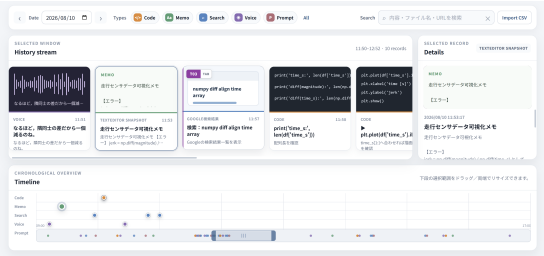

図 5　履歴閲覧ブラウザ

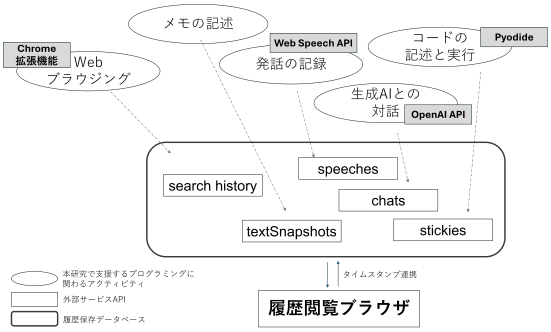

図 6　システム構成

れているのかといった，異なる作業間の時間的な関係を確認できる．タイムライン上の履歴を選択すると，中央の履歴一覧でも対応する履歴が選択され，その詳細を確認できる．

4　実装

4.1　システム構成

本システムは，HTML，CSS および JavaScript を用いて実装された，Web ブラウザ上で動作するクライアントサイド型のシステムである．3 章で述べた各ツールは，それぞれが独自の HTML 文書，ユーザインタフェースおよび JavaScript による処理を持つ独立した Web アプリケーションとして実装されており，必要に応じて個別のウィンドウで起動される（図 6）．ユーザによる入力やクリック，ドラッグ，コードの実行，音声認識の開始と終了，生成 AI へのプロンプト送信といった操作は JavaScript のイベントリスナによって取得し，イベントに対応する処理によって画面上の状態を変更するとともに，必要に応じてその時点のデータを履歴として生成する．すなわち本システムにおける履歴情報は，画面を定期的にキャプチャして生成するのではなく，各ツールにおいてあらかじめ定められた操作や状態変化を契機として生成される（表 1）．

各ツール間で JavaScript オブジェクトを直接受け渡したり，ウィンドウ間で常時メッセージを送受信したりはしない．各ツールがそれぞれ共通データベースにデータを書き込み，履歴閲覧ブラウザが後からそれを読み出す．このため，共通データベースはデータの保存先であると同時に，独立して動作する各ツールを接続するための共有領域としても機能する．

4.2　共通データベースと履歴の記録

作業履歴の保存には，Web ブラウザが提供するローカルデータベースである IndexedDB を使用し，そのアクセスには JavaScript ライブラリである Dexie.js を用いている．IndexedDB は Web ブラウザのオリジンごとに管理されるため，本システムを構成する各ツールは，同一のオリジンから起動し，同じ名称とバージョンの共通データベースを参照する構成としている．共通データベース内では，ツールごと，またはデータの種類ごとにオブジェクトストアを分けている（表 1）．各オブジェクトストアに保存されるデータの項目は異なるが，いずれも履歴を識別するための ID と，履歴が生成された時刻を示す情報を持たせている．

履歴データの ID には，Dexie.js および IndexedDB によって自動的に増加する主キーを使用する場合と，各ツール内で生成したオブジェクト ID を使用する場合がある．前者ではスキーマ上で`++id` を指定する．一方で，二次元平面上のオブジェクトやチャットなど，同一のデータを継続的に更新する必要がある場合には，ツール内で使用している識別子を保存し，その識別子を用いて既存のデータを検索および更新する．新しい履歴を追加する場合には `add` を用い，既存のオブジェクトの状態を上書きする場合には `put` を用いる．

本システムでは，各ツールが「現在の作業状態を復元するためのデータ」と「その時点の状態を履歴として残すデータ」とを区別して保存する．メモツールでは，入力のたびに文章全体を履歴として追加すると

データ量が大きくなり，履歴閲覧ブラウザ上にも内容のほとんど変わらない履歴が多数表示される．そのため，通常の編集状態を保持する処理と，閲覧対象となるスナップショットを生成する処理とを分けており，履歴閲覧ブラウザへ渡される履歴の単位は，キー入力単位ではなく，ユーザが明示的に保存した時点の文章全体となる．コードツールでは，セルの編集や移動のたびに画面上の最新状態を上書きする一方，実行時にはその時点のコード，メモ，実行結果および位置を履歴として追加する．AI 対話ツールでは，プロンプトオブジェクトの編集を同一オブジェクトの更新として扱い，送信操作が行われた時点のプロンプトを，変更されない履歴として別に保存する．これにより履歴閲覧ブラウザでは，単に現在表示されているプロンプトではなく，生成 AI に対して実際にどのような内容が送信され，どのような応答が返されたかを表示できる．

4.3 各ツールにおける履歴生成の実装

発話ツールでは，ブラウザ上で音声認識を行うために Web Speech API を使用する．開始ボタンが押されると，JavaScript から SpeechRecognition またはブラウザが提供する互換実装を初期化し，音声認識処理を開始する．Web Speech API から返される認識結果には，認識途中の暫定結果と，認識内容が確定した最終結果が含まれるため，イベントハンドラ内で両者を区別して処理する．暫定結果は現在認識されている内容を画面上に表示するためにのみ使用し，最終結果が返された場合には，確定した発話テキストを確定時刻とともに履歴データとして生成し，発話履歴用オブジェクトストアへ追加する．終了ボタンが押された場合，または Web Speech API から終了イベントが返された場合には，認識処理を停止し，画面上の認識中状態を解除する．

AI 対話ツールは，チャットインタフェースから OpenAI API を呼び出す構成としている．送信イベントの処理内で，その時点のプロンプト文字列，利用するモデル，それまでの対話に含まれるメッセージなどを取得し，OpenAI API が受け取るリクエスト形式に変換して送信する．リクエスト送信前には送信時点のプロンプトを履歴として共通データベースへ保存するため，プロンプトオブジェクトが後から編集された場合でも，実際に API へ送信された時点の内容を残すことができる．応答が返された場合には，レスポンスとして得られた JSON データを解析して生成されたテキストを取り出し，プロンプト ID，チャット ID，応答テキスト，応答を受信した時刻などを含むデータを，送信したプロンプトと関連付けて保存する．API からエラーが返された場合には，正常な応答として保存せず，画面上にエラー内容を表示して処理を終了する．生成 AI との複数の対話はチャット ID によって分けて管理し，過去のチャットを選択した場合には，対応するデータを共通データベースから取得し，作成時刻または配置情報に基づいて画面を復元する．

コードツールと AI 対話ツールでは，二次元平面上のオブジェクトの位置や大きさもデータに含めて保存する．ドラッグ操作中はポインタ座標を取得してオブジェクトの CSS 上の位置を更新し，操作が終了した時点で変更後の座標をオブジェクトのデータへ反映して共通データベースを更新する．これにより，ツールを再度起動した際には，保存されている位置情報を読み出し，各オブジェクトを以前と同じ位置に再配置できる．この位置情報は履歴閲覧ブラウザの時系列表示そのものには必須ではないが，各ツール上の作業状態の復元と，過去のワークスペースの再現のために保存している．なお，メモツールにおける過去状態の復元処理は，スナップショットの取得，現在内容の退避，入力欄の置換，画面表示の更新という順序で実行する．自動退避用のデータは常に一つだけ保持し，新たに復元を行うたびに更新する．

4.4 Web 検索履歴の収集

Web 検索履歴は，本システムとは別に実装した Google Chrome の拡張機能によって収集する．検索履歴収集の開始操作が行われると，収集中であることを示す状態を保持し，その後に Chrome 上で発生する検索，ページの表示，タブの生成，タブの切り替えなどのイベントを取得する．収集終了操作が行われると，それ以降に発生したイベントは記録対象か

ら除外する．検索が実行された場合には，表示された検索結果ページの URL や検索欄の状態から検索キーワードを取得し，イベントの発生時刻，ページタイトル，URL，タブ ID およびタブのタイトルなどとともに保存する．同じ検索結果ページ上で検索語が変更された場合にも，変更後の URL または検索状態を解析し，新たな検索イベントとして扱う．

収集された履歴は，収集終了後に CSV 形式で出力する．CSV の各行には，一つのイベントに対応する時刻，イベント種別，タイトル，URL，タブ ID，タブタイトルおよび検索キーワードなどが格納される．この CSV ファイルを本システムへインポートすると，JavaScript によって各行を解析し，列名と値を対応付けたオブジェクトへ変換したうえで，共通データベース内の `searchHistory` オブジェクトストアへ追加する．Chrome 拡張機能が共通データベースへ直接書き込むのではなく CSV を介する構成としているため，検索履歴の収集処理と本システムのデータベース処理は分離されている．この構成では，収集した CSV ファイルを確認してから共通データベースへ取り込むことができる一方，検索イベントの発生直後に履歴閲覧ブラウザへ反映されるわけではない．

4.5 履歴の統合と時刻の正規化

履歴閲覧ブラウザは，起動時に Dexie.js を用いて共通データベースを開き，コード，テキスト，発話，生成 AI との対話および検索履歴に対応する各オブジェクトストアからデータを取得する．複数のオブジェクトストアに対する取得処理は非同期に実行され，すべての取得が完了した後に，取得したデータを一つの配列へ統合する．

各オブジェクトストアから取得されるデータは，そのままでは項目名や構造が異なる．例えば時刻は，検索履歴では `visitTime` に保存されているが，他の履歴では `timestamp` や `createdAt` などに保存される．また，内容もコードと実行結果，認識された発話テキスト，プロンプトと生成 AI の応答と様々である．そこで，取得元のオブジェクトストアごとに変換処理を定義し，各履歴から，履歴の種類，発生時刻，表示用のタイトル，表示する内容，元データの ID および必要に応じた付加情報を抽出して，履歴閲覧ブラウザ内で扱う共通の履歴オブジェクトへ変換したうえで，すべてを同じ配列に追加する．

時刻の値が文字列として保存されている場合には，JavaScript の Date オブジェクトまたは数値のタイムスタンプへ変換して比較し，履歴ごとに時刻の表記形式が異なる場合には，それぞれの変換処理内で正規化する．時刻として解釈できないデータや時刻情報を持たないデータについては，表示対象から除外するか，別の基準によって末尾へ配置する．この時刻の正規化処理によって，異なるツールが異なる方法で保存した履歴であっても，同一の時間軸上に配置できる．すなわち本システムでは，各オブジェクトストアのデータ構造を完全に共通化するのではなく，各ツール固有のデータ構造を維持したまま，履歴閲覧ブラウザ側で表示に必要な情報を抽出する方式を採用している．

時刻順に並べた履歴は，JavaScript によって DOM 要素へ変換し，履歴の種類ごとに表示処理を分けて，履歴閲覧ブラウザのタイムライン領域へ追加する．画面上での配置位置は，元のオブジェクトストアではなく，共通の発生時刻によって決定される．なお履歴閲覧ブラウザは，個々のツールから履歴を受信して逐次描画するのではなく，共通データベースに保存されているデータを読み出した時点でタイムラインを構築する．したがって，履歴閲覧ブラウザを起動したまま別ウィンドウで新しい履歴が追加された場合，そのデータを画面へ反映するためには，再読み込みや更新操作によってデータベースを再度読み出し，タイムラインを再構築する必要がある．

5 提案システムの利用シナリオ

本章では，本提案システムを用いてプログラミングを行う例を示す．ここでは，ユーザが自動車の走行中に取得された加速度センサのデータを可視化する作業を想定する．データは CSV 形式で保存されており，各時刻における x 軸，y 軸，z 軸方向の加速度が記録されている．ユーザは Python を用いた基本的なプログラミングは経験しているが，分からないことについては Web 検索や生成 AI を利用しながら作業を進める．

ユーザはまず，メモツールに,「CSV を読み込み，x，y，z の 3 軸をグラフとして表示する」と実装方針を書き留める．その後，コードツールで CSV を読み込み，各軸の値を時間に沿って描画するコードを作成する．書き方が分からない部分については Web で調べながらコードを修正し，3 軸の加速度をグラフとして表示することができた．

次にユーザは，3 軸それぞれの値だけでなく，自動車の動きが大きく変化した箇所も見てみたいと考える．そこで，加速度の大きさを求め，さらにその変化を計算するコードを追加する．変化を求めるために np.diff を用いてグラフを描こうとしたところ，x 軸と y 軸のデータ数が一致していないことを示すエラーが発生する．

ユーザはエラーの原因が分からず,「なんで 41 と 40 になってるんだろう」と発話する．その後，エラーメッセージを Web で検索したり，生成 AI にエラーの意味を尋ねたりする．また，コードを変更してそれぞれのデータ数を表示し，時刻のデータは 41 個，計算した値は 40 個であることを確認する．試した内容や分かったことについてはメモツールにも書き留める．

しばらく調査を続けるうちに，ユーザは検索したページや試したコードが増え，どこまで原因が分かっていて，次に何を確認すればよいのか分からなくなる．そこで一度コードの記述を離れ，履歴閲覧ブラウザを開く．

一日のタイムラインを見ると，それまで比較的まばらであった履歴点が，エラーが発生した 11 時半頃から急に密集している．ユーザはこの部分を選択し，種類ごとに分けられた履歴を見る．そこには，コードの実行，Web 検索，発話，生成 AI との対話，メモが短い間隔で並んでいる．

ユーザがそれらを順に確認すると，最初にエラーが発生した後,「データ数が合わない」という内容を検索し，生成 AI にも同じエラーについて質問していたことが分かる．その後，コード上でデータ数を確認し，時刻が 41 個であるのに対して np.diff の結果が 40 個になっていることを確かめていた．さらに,「np.diff だと一つ減るらしい」というメモや発話も残されている．

これらの履歴を見返すことで，ユーザは，問題が CSV の読み込みにあるのではなく，np.diff を用いた後にデータ数が一つ減っていることにあると，それまでの作業を整理する．そこで生成 AI との対話に戻り,「元の時刻データと同じ数のまま変化を求める方法はあるか」と質問する．その結果として np.gradient を用いる方法を知り，コードツールで試してみる．今度は時刻と計算結果のデータ数が一致し，グラフを描画することができた．

エラーへの対処を終えた後も，ユーザは履歴閲覧ブラウザ上でしばらく履歴を眺める．その中で，以前に別のセンサデータを可視化した際のコード履歴が目に留まる．そのコードでは，細かく変動する値をそのまま描画するのではなく，移動平均を求めてグラフを滑らかにしていた．この履歴は，今回発生したエラーを解決するために探していたものではなかったが，ユーザはその内容を見て,「今の加速度のグラフにも使えるかもしれない」と考える．

6　まとめと今後の課題

本稿では，プログラミングに伴う多様な作業を支援・記録し，そこで生じた履歴を横断的に閲覧するプログラミング支援環境について，そのデザインと実装を報告した．本システムでは，コードの記述・実行や生成 AI との対話を行う Dialogue Tools，テキストメモや発話を記録する Monologue Tools，および，これらの履歴と Web 検索履歴を同一の時間軸上で閲覧する履歴閲覧ブラウザを構築した．

本システムでは，個々の履歴を後から確認・再利用するだけでなく，異なる種類の履歴を時間的な関係の中で捉えることで，過去の作業の流れやその際の思考を振り返ることを目指している．履歴を閲覧する過程で，当初探していたものとは異なる過去の履歴に触れ，それを現在のプログラミングに利用するような使い方も想定している．また，コードやプロンプトを二次元平面上に配置しながら作業する形をとることで，作業中に考えを展開するための材料として働いた配置が，そのまま後から当時の考えを想起する手がかりとしても残るようにしている．

今後は，実際のプログラミング作業において本システムを継続的に利用し，どのような履歴がどのような場面で参照され，現在の作業にどのようにつながるのかを調査する．長期間の利用によって蓄積される多数の履歴の中から，現在の作業に関連する履歴や，ユーザ自身が意識していなかった過去の履歴との出会いをどのように支援するかについても検討する．あわせて，各ツールで新たに生成された履歴を履歴閲覧ブラウザへ逐次反映するなど，作業中の履歴閲覧をより円滑にするための実装上の改善を進める．また，Tantithamthavorn が指摘しているように，AI coding agent が実装や検証の多くを担うような開発形態 (Agentic Software Engineering) では，プログラマの作業の比重がコードの記述から仕様の具体化や結果の検証へ移るとされる [6]．そこで残すべき履歴が何であり，どの粒度で記録することが振り返りに有効なのかについても検討していく必要があると考える．

参 考 文 献

[1] Chen, V., Zhu, A., Zhao, S., Mozannar, H., Sontag, D. A., and Talwalkar, A.: Need Help? Designing Proactive AI Assistants for Programming, *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, CHI '25, New York, NY, USA, Association for Computing Machinery, 2025, pp. 881:1–881:18.

[2] Cockburn, A. G.: Reflexive CSCW and Managing Commitments, *IEE Colloquium on CSCW: Some Fundamental Issues*, 1991, pp. 1–5.

[3] Hill, W. C., Hollan, J. D., Wroblewski, D., and McCandless, T.: Edit Wear and Read Wear, *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '92)*, 1992, pp. 3–9.

[4] Nakakoji, K. and Yamamoto, Y.: What Does the Representation Talk Back to You?, *Knowledge-Based Systems*, Vol. 14, No. 8(2001), pp. 449–453.

[5] Sáenz, J. P. and Russis, L. D.: Dear Diary: On Documenting Novices' Development Process, *2022 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2022, pp. 1–3.

[6] Tantithamthavorn, K.: *Agentic Software Engineering: A Practical Guide for the AI-Native Engineer*, 2026. Accessed: 2026-07-31.

[7] Yamamoto, Y., Nakakoji, K., and Takada, S.: Hands-on Representations in a Two-Dimensional Space for Early Stages of Design, *Knowledge-Based Systems*, Vol. 13, No. 6(2000), pp. 375–384.

[8] Yen, R., Zhao, J., and Vogel, D.: Code Shaping: Iterative Code Editing with Free-form AI-Interpreted Sketching, *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, CHI '25, New York, NY, USA, Association for Computing Machinery, 2025, pp. 872:1–872:17.

[9] Zamfirescu-Pereira, J. D., Jun, E., Terry, M., Yang, Q., and Hartmann, B.: Beyond Code Generation: LLM-supported Exploration of the Program Design Space, *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, CHI '25, New York, NY, USA, Association for Computing Machinery, 2025, pp. 153:1–153:17.

[10] 渡部丈, 中小路久美代: 知的創造活動における「過去の自分との対話」—自己の履歴情報が喚起する記憶と想起の理論的位置づけ—, 日本認知科学会第 *43* 回大会発表論文集, 2026. 発表予定.

[11] 渡部丈, 山本恭裕, 中小路久美代: プログラミングに伴う多様な活動の履歴をブラウジングする環境 HALO-Browser のデザインと試作, 情報処理学会研究報告, Vol. 2023-HCI-202, No. 5, 2023, pp. 1–7.

[12] 渡部丈, 山本恭裕, 中小路久美代: 自身の作業履歴を活用することの記号論的解釈の試み—プログラミング支援を題材として—, 人工知能学会全国大会論文集, 2024, pp. 1–4. 1N3-GS-9-01.