エンジニアリングデザイン
レポート課題

C言語学習プラットフォーム　くもんCき

2014年1月25日

2009SE298　脇田宏威
2011SE188　中島啓貴
2011SE194　成田貴大

1. 要求分析

1.1. 現状把握

多くの機関や企業が情報システムを導入しICT化を図ることが当たり前となった現代，政府機関で義務教育にプログラミングを導入する案が話し合われたりと，ますますプログラミングは我々にとって一般的なスキルとなり，同時にその重要性は高まっている．しかし，プログラミング言語は「言語」と呼ばれるものの我々が普段利用する自然言語との隔たりは大きい．また普段から意識する場面が少なく生活から身に付くということも期待できないため能動的な学習が必要となるが，学習を始めるためには環境構築といった大きなハードルを越えなければならない．

ようやく学習環境を得たプログラミング初学者の多くが初めに直面する問題は大量のエラーである．このような状況では，記述の全てが間違っているのか，それとも一部の間違いが大量のエラーを誘発しているのか分からず，過酷なデバッグ作業を強いられることとなり，最終的に挫折してしまうことが想像に難くない．

そこで我々は，煩雑な環境構築なしに利用でき，エラーを初学者にも対処できるレベルにまで分割できる学習環境を提供することで，プログラミング初学者の学習効率やモチベーションを改善することができるのではないかと考えた．

1.2. ターゲットの設定

プログラミング言語の学習環境を提供するにあたって，最も多くのプログラミング言語のベースとなっているC言語の問題を提供することにした．この選択は初めてプログラミング言語に触れる人をターゲットにするとともに，グローバルに需要があるが，この方法が最も効果的であるという学習法が確立されていないためである．

プログラミング言語の学習において試行錯誤を繰り返しエラーに対処することは能力の向上につながる．そのため，特にこの効果が現れると考えられるプログラミング言語に触れ始めた初心者から，ある程度のプログラムなら読み書きできるといった中級者までをターゲットとして設定した．

1.3. 解決案の比較

プログラミング言語学習プラットフォームの実現方法を検討したところ，以下の5つが挙げられた．

- 公文式を模した問題演習プラットフォーム
- よくある入門サイトを踏襲した学習サイト
- 中級者向けコードのリーディングサイト
- いわゆる知恵袋を踏襲した相談サイト
- ヴィジュアルプログラミングプラットフォーム

ここで，それぞれの実現方法についてメリットとデメリットを挙げて詳しく説明する．

公文式を模した問題演習プラットフォーム

公文式を模した，小規模問題に繰り返し解答できるプラットフォームを作成する.

公文式では自主学習プリントを用いて学習を行う．自主学習プリントには単純な計算問題などが出題されており，各自で学年に関係なく好きなところから始められるようになっている.繰り返し挑戦して自分の苦手な単元を割り出したり，苦手な単元の少し前の部分から復習学習を行ったりすることができる.

メリット

- 問題解決の達成感を断続的に得られ，モチベーション維持に繋がる.
- 問題規模が小さいためエラー箇所が特定しやすく，初学者でも十分対応できる.

デメリット

- 繰り返し学習に利用できる程度の量の問題を作成する必要がある.

よくある入門サイトを踏襲した学習サイト

Web上に存在する「○○入門」といったサイトの内容をそのまま中級者向けに作成する.

メリット

- 問題解決の達成感を断続的に得られ，モチベーション維持に繋がる.
- 問題規模が小さいためエラー箇所が特定しやすく，初学者でも十分対応できる.

デメリット

- 効果的な課題設定や解説が困難.

中級者向けコードのリーディングサイト

オープンソース・ソフトウェアのソースコードは初学者や中級者にとって高度過ぎるため,より基礎的な内容のまとまったソースコードを閲覧できるサイトを作成する.

メリット

- プログラムは読むためのものであるという点を重視できる.
- 複雑な処理構造の参考にでき，学習効果を期待できる.
- オープンソース・ソフトウェアにハードルを感じていた利用者に効果が期待できる.

デメリット

- 有益で複雑過ぎないソースコードの提供が困難.

いわゆる知恵袋を踏襲した相談サイト

プログラミングに関する相談に特化した相談サイトを作成する.

メリット

- 検索にヒットしにくい問題に対して効果的に解法を提供できる.
- 解決例をシェアすることで同様の問題に直面した利用者にも役立つ.

デメリット

- 相談に答える人員が必要.
- 相談者が既存の解決例を確認しない可能性が高い.
- 既存の類似サービスが多数存在する.

ヴィジュアルプログラミングプラットフォーム

ヴィジュアルプログラミングのできるプラットフォームを作成する.

メリット

- アルゴリズムや文法の意味，動作を効率的に学習できる.
- 利用者の負担が少ない.

デメリット

- 設計，デザインが困難.
- エラーの起こらないような設計では達成感を得にくい.

ここまで挙げてきた各実現方法について検討した結果，プログラミング言語学習において繰り返し学習が効果的であったという経験則や，問題規模の小ささによるテンポの良さ，エラーと原因の対応が把握しやすいという特徴から，「公文式を模した問題演習プラットフォーム（以下，本プラットフォーム）」を実現方法として採用した.

運用体制の検討

本プラットフォームを実現するにあたって，どのような運用体制でサービスを提供するか検討する．このようなシステムを運用する場合，システムそのものの管理は管理者が行うが，問題提供については管理者によって提供される場合と利用者が投稿する場合の2通りが考えられる．これら2つの場合について，メリットとデメリットをまとめた表を以下に示す.

	メリット	デメリット
管理者が提供	問題毎に意図を込められる	問題数に限界がある
利用者が投稿	大量の問題を提供できる	問題の質にばらつきが出る 問題の管理が煩雑になる

本プラットフォームの繰り返し学習という性質上，大量の問題を提供できることは大きなメリットと言えるため，利用者が問題を投稿する運用体制を採用した.

提供方法の検討

本プラットフォームを実現するにあたって，どのようなメディアでサービスを提供するか検討したところ，不特定多数から問題投稿を得る必要があるため，Webアプリケーションとしての提供を採用した．これにより，端末やOSに依存せず本プラットフォームを利用できるため，より多くの問題を提供できると見込める.

具体的にサーバーはどこまでサービスを提供するかの決定

まず，利用者による問題投稿機能を提供する場合，投稿者の負担軽減や正確さの保証のため自動採点機能の提供が必須と言える．自動採点機能の実現方法を検討したところ，以下の5つが挙がった.

- サーバ上でコンパイルして実行結果を比較する
- 外部サービス（ideone.com など）を利用した実行結果を比較する
- クライアント上のインタプリタで実行結果を比較する
- クライアント上での構文解析結果を比較する
- ソースコードを比較する

これらの実現方法それぞれについて，メリットとデメリットを列挙した表を以下に示す．

	メリット	デメリット
サーバ上でコンパイル	様々な解答に対応可能	サーバの負担が大きい
外部サービスの利用	様々な解答に対応可能 実現が容易	細かい設定ができない
クライアント上のインタプリタ	様々な解答に対応可能 サーバの負担が小さい	実現が困難
クライアント上での構文解析	サーバの負担が小さい	様々な解答に対応不可能 実現がやや困難
ソースコードを比較	実現が容易	様々な解凍に対応不可能

表にある「様々な解答に対応」というのは，プログラミング特有の要求である．プログラミングにおいて，同一の出力を得るためのソースコードは1通りには定まらないが，プログラミングの目的は同一のソースコードを作成することではなく，同一の出力を得ることであるため，ソースコードの不一致は本来不正解として扱われるべきではないと言える．

表をもとに検討を行った結果，様々な解答に対応できない「クライアント上での構文解析」と「ソースコードの比較」は解決方法として適切ではないと判断し，また「クライアント上のインタプリタ」は開発期間と実現難易度を考慮した結果不採用とした．「サーバ上でコンパイル」と「外部サービスの利用」の2案については簡単なサンプルプログラムを作成し，実際の動作からそれぞれ以下に列挙する特徴が明らかとなった．

「サーバ上でコンパイル」の特徴

- 実装が比較的容易である
- 通常よりセキュリティを考慮する必要がある
- gccのエラーメッセージをそのまま表示できる

「外部サービスの利用」の特徴

- 一部のエラーメッセージが表示されない
- エラーメッセージが丁寧だがgccと異なる
- ファイルの入出力を実行できない

「サーバ上でコンパイル」の実装はセキュリティを考慮しなければ，当初の想定より簡単であった．セキュリティについては，現段階ではテスト用としてBASIC認証を用いたアクセスのみを許可しているため，考えないものとする．

解答者は本システムによって表示されるエラーメッセージから，メッセージ内容とエラー原因の対応を学習することが期待されるため，gccのエラーメッセージが表示できるという点は，本システムによってgccのエラーメッセージを利用したデバッグ能力を向上させられることを意味する．

ファイルの入出力はプログラミングにおいて基本的な要素の1つといえるため，これに関連した問題を出題できないことは大きなデメリットであるといえる．

ここまでの内容を総合的に検討した結果，「サーバ上でコンパイル」を採用することに決定した．

利用者による問題検索方法について

利用者が問題に解答するにはまず，利用者の解きたい問題を見つけられる必要がある．そのためには問題に対し指標となるものを付加する必要があると考えられた．

本プラットフォームでは各問題のタイトルに加えて，複数のタグを付加できる機能を実現することでこの問題を解決することとした．これにより，タイトルによる検索だけでなくタグによる検索も可能となり，自分の解けなかった問題と同じタグの問題を検索することで，利用者にとって最も有益な問題のみを解き続けることが可能となる．

2. 設計

2.1. 要求仕様書

- システムは投稿者に投稿フォームを送る．
- 投稿者はシステムに問題データを送る．
- システムは投稿者に投稿結果（成否）を送る．
- システムは問題データベースに問題データを登録する．
- システムは問題データベースから問題データを取り出す．
- 解答者はシステムに問題リクエストを送る．
- システムは解答者に解答フォームを送る．
- 解答者はシステムに解答データを送る．
- システムは解答者に解答結果を送る．

2.2. プロセス仕様書

この要求仕様を元にプロセス仕様書の記述を行った．

- 投稿者は問題受理係に投稿リクエストを送る．
- 問題受理係は投稿リクエストを受け取ったら，投稿者に投稿フォームを送る．
- 投稿者は投稿フォームを介して問題受理係に問題データを送る．
- 問題受理係は問題データを受け取ったら，小問題データから実行リクエストを生成し，実行係に実行リクエストを送る．
- 問題受理係は実行係から実行結果を受け取ったら投稿者に投稿結果フォームを送り，投稿確認を行う．
- 投稿確認が受諾された場合，問題受理係は問題データベースに問題データを登録する．

- 解答者は問題一覧係に問題一覧リクエストを送る．
- 問題一覧係は問題一覧リクエストを受け取ったら，データベースから問題データを取得し，解答者に問題一覧を送る．
- 解答者は出題係に問題リクエストを送る．
- 出題係は問題データベースから問題データを取得する．
- 出題係は解答者に解答フォームを送る．

- 解答者は解答フォームを介して解答受理係に解答データを送る．
- 解答受理係は解答データから実行リクエストを生成し，実行係に実行リクエストを送る．
- 解答受理係は実行係から実行結果を受け取ったら問題のテストパターンと照合し，解答者に解答結果を送る．

- 実行係は，実行リクエストを受け取ったら，実行結果を送り主に返す．

アンダーラインが入っているのもの以下のデータ辞書で定義する

2.3. データ辞書

- 問題データ　＝ 問題ID + 1{ 小問題データ } + { タグ }
- 小問題データ　＝ [穴埋め問題データ]
- 解答データ　＝ 問題ID + 1{ 小解答データ }
- 小解答データ　＝ [穴埋め解答データ]

- 穴埋め問題データ　＝ 問題文 + ソースコード + 1{ 穴データ } + 1{ テストパターン }
- 穴データ　＝ 穴ID + 開始位置 + 終了位置
- テストパターン　＝ （テスト入力） + 期待される出力

- 穴埋め解答データ　＝ 1{ 穴ID + 内容 }

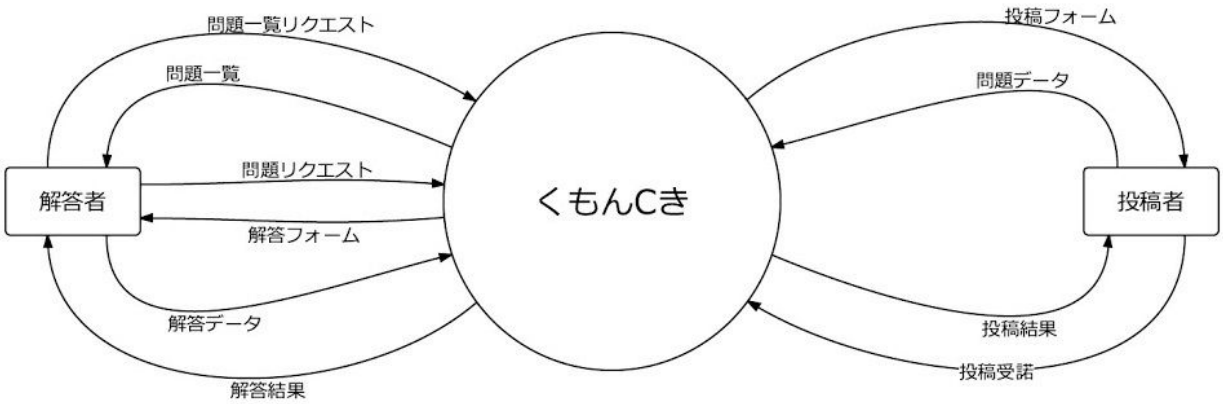
- 問題リクエスト　＝ 問題ID
- 投稿結果　＝ 実行結果 + 投稿成否
- 解答結果　＝ 実行結果 + 解答成否

- 実行リクエスト　＝ ソースコード + 1{ テストパターン } + 送り主
- 実行結果　＝ コンパイラ出力 + 1{ 標準出力 }

データ辞書に関して今回は実装する問題の型は穴埋め問題のみとしたが，今後の拡張として選択問題などの他の問題型の追加を想定しデータ辞書の記述を行った.

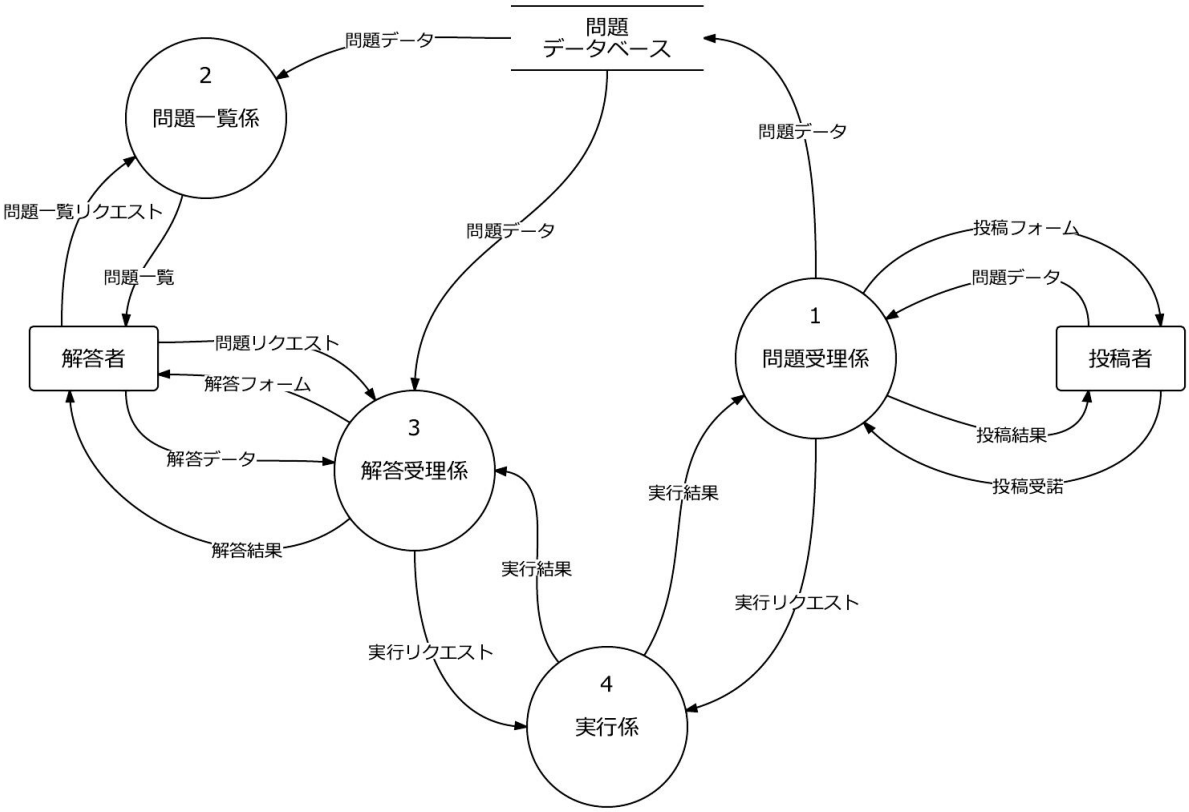
コンテキスト図

プロセス仕様とデータ辞書を元にコンテキスト図の記述を行った.



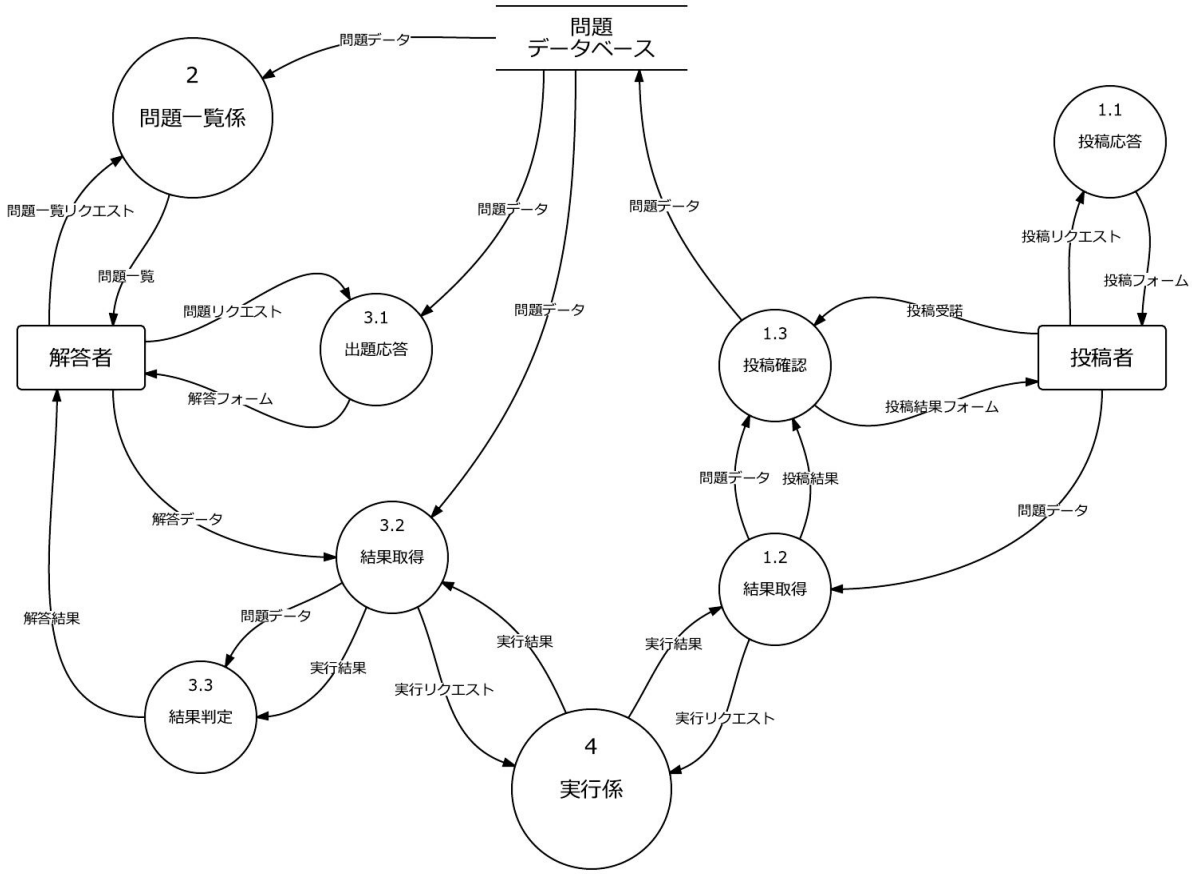
DFD Level1

コンテキスト図を元にDFDの記述を行った.



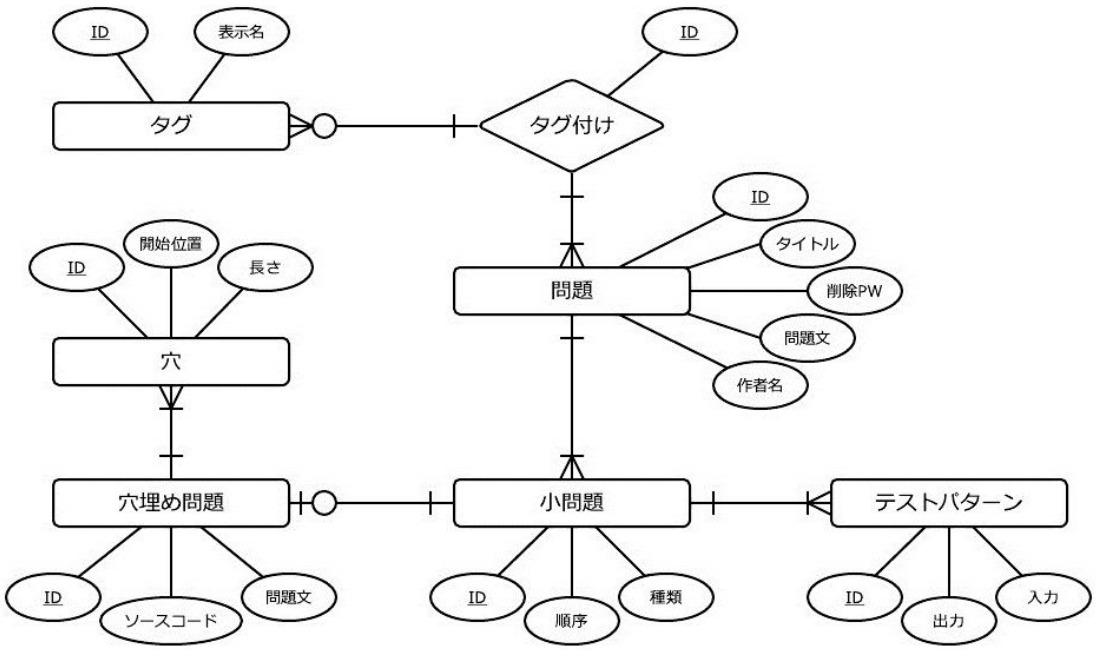
DFD Level2

DFDを分割分割し，より詳細なDFDの記述を行った.



ER図

DFDとデータ辞書よりDBで取扱うデータに関して書き出しを行い正規化しのER図を記述を行った.



クラス図

開発を進める過程で，データベース上の問題データとHTTP POSTデータ上の問題データの相互運用性を確保するため，ER図を基に以下のクラス図を作成した.

3.実装

3.1. 本プラットフォームを構成する技術

本プラットフォームでは以下に挙げる技術を利用している.

- HTML
- CSS
- PHP
- PostgreSQL
- JavaScript

要求定義でも述べた通り，本プラットフォームはWebアプリケーションとして実現するため表示にはHTMLとCSSを用い，主な処理にはHTMLとの親和性の良さからPHPを採用した．尚，一部の処理におけるUX向上のためJavaScriptも用いている．データの保存には検索性やPHPとの連携の良さからPostgreSQLを採用した.

以降に，特筆すべき実装のポイントをまとめる.

3.2. 問題データの通信

本プラットフォームではUX向上のため，投稿ページと解答ページのUIに類似性を持たせている．このため，両者が出力の際利用するデータ構造を共通化することで，両者の出力処理も共通化することができ，開発効率や保守性も向上できると考えた．またUI出力処理を記述する際，将来的に穴埋め問題以外の問題種類もサポートすることを考慮すると，大量の分岐処理により保守性が著しく損なわれることも予想される．これらの理由から，本プラットフォーム動作中に扱うデータをオブジェクト指向に基いたデータ形式に統一し，ポリモフィズムを用いて出力処理の保守性の向上を図った．これにより，当初の構造化手法を用いた設計をオブジェクト指向設計に変更した.

3.3. 穴埋め問題作成時のUI

問題の投稿に関してユーザが複雑な操作を介することなく希望通りの問題作成が可能となるUIを心がけ設計を行った.

実装の初期の段階ではPHPを積極的に使いJavaScriptを用いることなく問題投稿のページを作成する設計としていたが，JavaScriptを用いることで動的なページを可能となり，大幅にUX改善されるということが調査から浮き彫りになったので設計を変更し，PHPとJavaScriptを併用した.

3.4. 全問正解ページにおけるユーザ解答並行表示UI

本プラットフォームの成否判定では投稿者と解答者でソースコードが異なっても正解と判定される場合がある．解答者が自身が解答したソースコードとした投稿者のソースコードを比較することが必要であり，この行為が学習の向上につながる．解答の平行表示を行うことで，比較することを実現できる.

穴埋め問題では穴である部分が比較対象となり，全問解答ページでは投稿者のソースコードを表示する．解答者の解答はJavaScriptにより穴である部分にマウスを置くことで吹き出しとして表示される．この機能により解答の平行表示を実現する.

4. テスト

現在想定しているテスト項目を以下に記述する.

投稿部門

小問題の問題数に関するテスト

- 小問なし
- 穴埋め問題
  - 1問だけ
  - 3問だけ

小問題1問についての実行に関するテスト

- コンパイルエラーなし
  - 実行時エラーなし
    - 入力なし
    - 入力1行
    - 入力3行
      - 出力なし
      - 出力あり
  - 実行時エラーあり
- コンパイルエラーあり

小問題1問についての穴に関するテスト

- 穴なし
- インライン穴1つ
- インライン穴複数
- ブロック穴1つ
- ブロック穴複数
- 混合

解答部門

問題数と表示に関するテスト

- 小問なし
- 穴埋め問題あり
  - 1問だけ
  - 3問だけ

穴埋め問題の表示に関するテスト

- 穴なし
- インライン穴1つ
- インライン穴複数
- ブロック穴1つ
- ブロック穴複数
- 混合

小問題の解答欄の入力関するテスト

解答が空のとき
正解のとき
不正解のとき

1問の小問題に穴が単数の場合の採点に関するテスト

入力がない
入力が正しい
入力が誤り

1問の小問題に穴が複数の場合の採点に関するテスト

すべて入力がない場合
すべて入力が正しい場合
すべて入力が誤りの場合
入力がない 正しいの混合
入力がない 誤りの混合
入力が正しい 誤りの混合
入力がない 正しい 誤りの混合

小問題数と正否に関するテスト

小問題数がないとき
1問
- 正解
- 不正解

3問
- 全未採点
- 全正解
- 全不正解
- 正解 未採点混合
- 不正解 未採点混合
- 正解 不正解 未採点 混合

全問正解表示と問題数に関するテスト

小問題なし
1問
3問

5. 考察

今後の問題形式の追加について

本プラットフォームの実現にあたって，問題種類は穴埋め問題のみとして実装したが，今後の問題種類追加を考慮した設計となっている．具体的にはクラスの構造として穴埋め問題は小問題から継承の形をとっており他の形式を追加する場合も小問題から継承を行ってクラスを作成することで本プラットフォームに導入できるようになっている．また，データベースの構造も複数の問題形式を1つの大問題で使えるようになっている．

今後導入を考えている問題形式についてまとめておく

- 選択問題
  投稿者がソースコード上に穴を定義するとともに選択肢を定義する．
  解答者は選択を選ぶことで解答を行なう事ができる．
  問題文で意図が伝えにくい場合等に有効である．
- 誤り訂正問題
  投稿者はソースコードに穴と誤ったコードを設定する．
  解答者は誤りを発見し訂正することで正しい出力を得られるようにする．

このような問題を導入することで穴埋め問題だけでは扱いにくい問題に関しても対応できる．また，選択問題を最初に出題し後に穴埋め問題を出題する等の問題の形式を組み合わせにより，問題文でどのような問題なのかを長々と説明することなく効果的に出題の意図を解答者に伝えることができると考えている．

ユーザプロファイル機能の導入について

今後考えられる拡張としてユーザプロファイル機能を導入し，利用者ごとに解答履歴や投稿履歴を管理できるようにすることを検討している．この拡張により以下の機能が実現可能となる．

- ゲーム要素
- 利用者ーザの傾向分析
- 問題難易度の自動評価

ゲーム要素とは，問題の解答履歴や投稿履歴を元にレベリングや実績解放などといった，他の利用者との競争要素のことである．これを導入することで，解答者，投稿者共にモチベーションの向上が期待できる．

利用者の傾向分析とは，利用者の解答履歴に基づいてその利用者が苦手とする問題を，おすすめ問題として提示する機能のことである．これにより，利用者はより効率的に自身の苦手を発見することができ，自身の苦手を知ることが苦手でも効果的な学習に取り組むことが可能となる．

問題難易度の自動評価とは，挑戦した解答者に設定されたレベルや試行回数，かかった時間などに基づいて問題自体の難易度を自動的に評価する機能のことである．これにより，解答者が問題を探す際の新たな目安を作ることができる．

ユーザプロファイル機能の導入により，上で述べたような多くのメリットが見込まれるが，利用者にはプロファイルの作成のために登録，ログインしてもらう必要がある．これらの手順は利用者数増加を妨げる一因と成り得るため，登録していない状態でも機能を限定して利用できるようにするなどの緩和策を講じる必要があるかもしれない．また，登録時にメールアドレス等の個人情報の入力を要求する場合，それ相応のリスクマネジメントが必要になることも念頭に置いておく必要がある．

投稿者の問題修正について

投稿者が内容を修正できるか否か，どの程度修正できるのかということについて考察する．内容の修正を可能としたときに現れる主な問題として，ユーザプロファイルとの整合性の問題が挙げられる．例えば，投稿済みのある問題をミスに気付いた投稿者が修正した場合，修正前に正解したことになっている利用者のユーザプロファイルはどうあるべきか，ということである．

問題の削除のみを可能とした場合，正解履歴からもその問題を削除すれば良いが，投稿者の側からすると，ミスした場合は再び1から問題を作成して投稿しなければならなくなるため利便性が低下する．一方，問題の内容も修正可能とすると，投稿者はいつでも問題を手直しできる反面，解答者の側としては昨日正解した問題が今日は全く違う問題になっているということも起こり得る．

この問題は利便性と整合性のトレードオフの関係となっており，実際の解決方法を決定するには既存の投稿サービスについて，詳しく分析し検討する必要がある．

悪意のあるソースコードが入力された場合について

悪意のあるソースコードへの対策として以下の2つが挙げられる．

- サンドボックスを導入する
- ファイル操作関数などの危険なキーワードをパターンマッチングで検出する

以上の2つのメリットとデメリットをまとめた表を以下に示す．

	メリット	デメリット
サンドボックス	網羅度が高い 安全に実行可能	負荷の高い処理の実行を禁止しない
パターンマッチング	実現が容易 段階的適用が可能	網羅度が低い 自由度が低下する

表から，サンドボックスは網羅度が高いことが分かり，サンドボックスを導入することで，本システムに関するセキュリティの対策を完遂することができる．しかし，負荷の高い処理を実行した時にその処理を禁止しないため，それを補完する手法として，パターンマッチングを用いる．これにより，負荷の高い処理（ループ等）を検出し，実行を禁止することができる．

スパム対策

想定されるスパムの種類は以下である．

- 問題文の中に学習に関係ないURLを含む．
- 機械的な連続投稿

プロファイルの導入を行えばスパム投稿を行なうユーザ検出してアカウントのを凍結するなどの対策を取ることができるため，スパム行為自体の抑制を図ることができる．

問題文中のURLに関しては正規表現を使用しURL自体の検出を行うことは可能だが問題に関連したものなのか否かの判別は非常に困難である．くもんCきの性質とし他の学習サイトでの学習を前提としているため外部へのリンク機能を禁止してしまうことは利便性を著しく低下させる恐れがある．

外部へのリンクを可能にし，かつ不適切なリンクを検出するための方法の例としてこのような方法が挙げられる．

1. 問題文内での外部サイトのURLの記述は許可する．

2. URLを押した時点でクッションページを表示する.
3. クッションページではリンク先のページのタイトルやサムネイルを表示する.
4. クッションページでは解答者がリンクの内容をみてこのリンクは不適切であると感じた場合に押すボタン設置しておき押された回数を保存する.
5. 一定量を超えた場合リンクに警告メッセージを表示する.

このような対策を講じればリンクの使用と不適切なリンクの検出を両立することができる，しかしユーザにはクリックの手間が増えるので導入に関しては慎重になるべきである.

連続投稿はサーバ全体の処理速度の低下や検索性能の低下などを引き起こすので対策を講じる必要がある．これらの状況に有効な対処として以下が挙げられる.

- 投稿時間から，機械的な連続投稿を検知する
- CAPTCHAを使用する

例えば，連続投稿として検知されたものにのみCAPTCHAを利用することで，通常の投稿者への負担を抑えつつ連続投稿を抑制することができる.

重複問題の投稿

重複問題とは，既に投稿されている問題と本質的に内容の同じ問題のことである．不特定多数のユーザが問題を投稿するシステムでは，類似した内容が投稿されることは十分考えられる．いくらかの類似した問題は類題として許容可能だが，許容量以上の問題に対しては問題検索性能の低下やストレージ容量の圧迫を防ぐため，何かしらの軽減策を講じる必要がある.

しかし別々のユーザからの投稿で問題文やソースコードが完全に一致することはまずあり得ないため，類似した問題の判別は容易ではないと言える．この問題に対して，以下のような解決策が挙げられる.

- 問題文やソースコードに対する類似度分析機能を作成，または利用する.
- ユーザからの報告機能を作成する.
- 投稿前にタグやタイトルを基に類似すると思われる問題をサゼッションする.

6. 分担

2009SE298　脇田宏威（100時間程度?）

- 解答ページ作成
- 投稿ページ作成
- 全問正解ページ作成

2011SE188　中島啓貴（100時間程度？）

- Webサーバ設定
- PostgreSQL設定
- 一覧ページ作成
- 問題検索
- 問題作成のJavaScript関連

2011SE194　成田貴大（50時間程度）

- ドメイン管理
- PostgreSQL設定
- 問題クラスの設計／実装
- 各ページ間の連携

全員

- 本レポート全体（誰かが書いて全員で添削）
- サーバ諸設定