オンライン児童音楽教育システム開発における

CI環境の構築

力武研究室　　情報システム工学科5年 関　風稀

指導教員 情報システム工学科 力武克彰

目次

[1研究背景 - 1 -](#_Toc411978926)

[1.1児童対象メロディ創作アプリケーションシステム - 1 -](#_Toc411978927)

[1.2 muphic-onlineの仕様 - 1 -](#_Toc411978928)

[1.3スクラム - 3 -](#_Toc411978929)

[2研究概要 - 6 -](#_Toc411978930)

[2.1研究の目的 - 6 -](#_Toc411978931)

[2.2継続的インテグレーション - 6 -](#_Toc411978932)

[2.3研究概要 - 7 -](#_Toc411978933)

[3システムの要件導出 - 8 -](#_Toc411978934)

[3.1開発工程の分析 - 8 -](#_Toc411978935)

[4システムの仕様 - 10 -](#_Toc411978936)

[4.1システムの構成 - 10 -](#_Toc411978937)

[4.2監視システムの仕様 - 10 -](#_Toc411978938)

[4.2.1変更検知機能の仕様 - 10 -](#_Toc411978939)

[4.2.2ソースコード取得機能の仕様 - 10 -](#_Toc411978940)

[4.3テスト機能の仕様 - 11 -](#_Toc411978941)

[4.4デプロイ機能の仕様 - 11 -](#_Toc411978942)

[5システムの要素技術 - 12 -](#_Toc411978943)

[5.1 監視システムの要素技術 - 12 -](#_Toc411978944)

[5.1.1 監視対象 - 12 -](#_Toc411978945)

[5.1.2 CIツール - 12 -](#_Toc411978946)

[5.1.3 CIツール稼働サーバ - 13 -](#_Toc411978947)

[5.2 テスト機能の要素技術 - 13 -](#_Toc411978948)

[5.2.1テストフレームワーク - 13 -](#_Toc411978949)

[5.3 デプロイ機能の要素技術 - 14 -](#_Toc411978950)

[5.3.1 PaaS (Platform as a Service) - 14 -](#_Toc411978951)

[6システムの実装 - 15 -](#_Toc411978952)

[6.1 システム実装の準備 - 15 -](#_Toc411978953)

[6.2監視システムの実装 - 17 -](#_Toc411978954)

[6.2.1変更検知機能 - 17 -](#_Toc411978955)

[6.2.2ソースコード取得機能 - 19 -](#_Toc411978956)

[6.3テスト機能の実装 - 20 -](#_Toc411978957)

[6.4デプロイ機能の実装 - 20 -](#_Toc411978958)

[7システムの考察 - 22 -](#_Toc411978959)

[7.1監視システムの考察 - 22 -](#_Toc411978960)

[7.1.1監視システムにおける仕様の充足について - 22 -](#_Toc411978961)

[7.1.2監視システムにおける実際の工夫点について - 22 -](#_Toc411978962)

[7.2テスト機能の考察 - 23 -](#_Toc411978963)

[7.2.1テスト機能における仕様の充足について - 23 -](#_Toc411978964)

[7.2.2テスト機能における実際の工夫点について - 23 -](#_Toc411978965)

[7.3デプロイ機能の考察 - 24 -](#_Toc411978966)

[7.3.1デプロイ機能における仕様の充足について - 24 -](#_Toc411978967)

[7.3.2デプロイ機能における実際の工夫点について - 24 -](#_Toc411978968)

[7.4 システム全体における要求の充足について - 25 -](#_Toc411978969)

[8今後の展望 - 26 -](#_Toc411978970)

[8.1システムの評価 - 26 -](#_Toc411978971)

[8.2テストケースの導入 - 26 -](#_Toc411978972)

[9おわりに - 27 -](#_Toc411978973)

[共同研究者の紹介 - 28 -](#_Toc411978974)

[謝辞 - 28 -](#_Toc411978975)

[参考文献 - 29 -](#_Toc411978976)

# 1研究背景

## 1.1児童対象メロディ創作アプリケーションシステム

　児童が楽しみながら表現力や想像力を養えるソフトウェアとして，「メロディ付き物語創作支援システム(muphic)」が亀谷学人氏，瀬戸敏文氏らによって開発された[1]．

muphicは物語（絵）の作成と作曲を行えるソフトウェアである．児童はまず物語を作り，次にその物語の雰囲気やストーリーに合うようなメロディを考えて作曲する．この2つの手順を通して，児童は表現力などを養うことが出来る．

　しかしながら，いくつかの問題点により，小学校での運用は現状では難しいという事が明らかとなった．

技術的な面としては，muphicはデスクトップアプリケーション(端末へのインストールが必要)であるという点である．端末へのインストールが学校側からの制限で困難の場合，導入自体が難しいという問題がまず挙げられる．また，導入に成功したとしても，コンピュータ室等の端末台数に限りがある限られた教室での使用や，実習を行う授業時間の不足等が挙げられた．

他の面としては出前授業(児童館等の子供が集まる施設での公開講座)を受けた児童がmuphicを使用する傾向において，物語制作機能では一定の方向性(物語等の意味合い)を持たせた作成を行うのに対し，作曲機能では音符を置けるだけ置く，模様に見立てて音符を置く等の傾向が多く見られた．これはmuphicの「表現力や創造力を養えるソフトウェア」を満たせているかという問題が挙げられた．

　前述の問題を解決する為，muphicのWebアプリケーション化[2]，ゲーミフィケーション(コンピュータゲームの特徴的要素を日常生活の中に応用する手法) 導入による音楽要素の強化が提唱された．

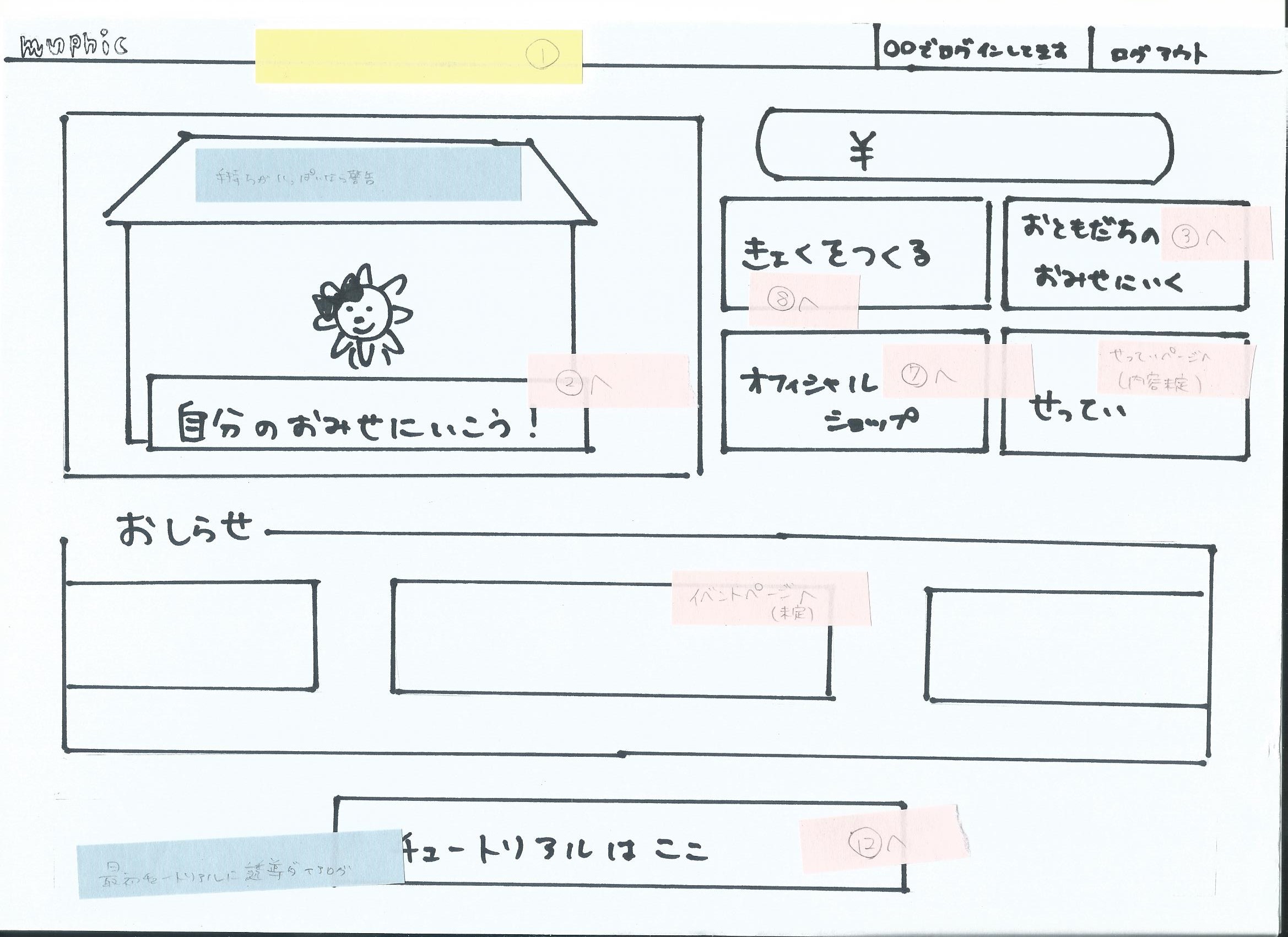
先行研究において，Webアプリケーション(muphic-online)の開発が始まった[3]．今年度のmuphic-online開発ではゲーミフィケーション要素を取り入れ，作曲機能のみに機能を絞り開発が行われた．

## 1.2 muphic-onlineの仕様

muphic-onlineは音楽の作曲を行う事が出来るゲーミフィケーション要素を取り入れたWebアプリケーションである．曲の作成，お店を用いてその曲を児童間で共有する，他の児童の作曲にコメントを入れる等を通して，児童は楽しみながら表現力や想像力を養う事が出来る．

muphic-onlineは主にメニュー画面，お店画面，作曲画面からなる．児童はそれぞれ個人のお店を持つ．

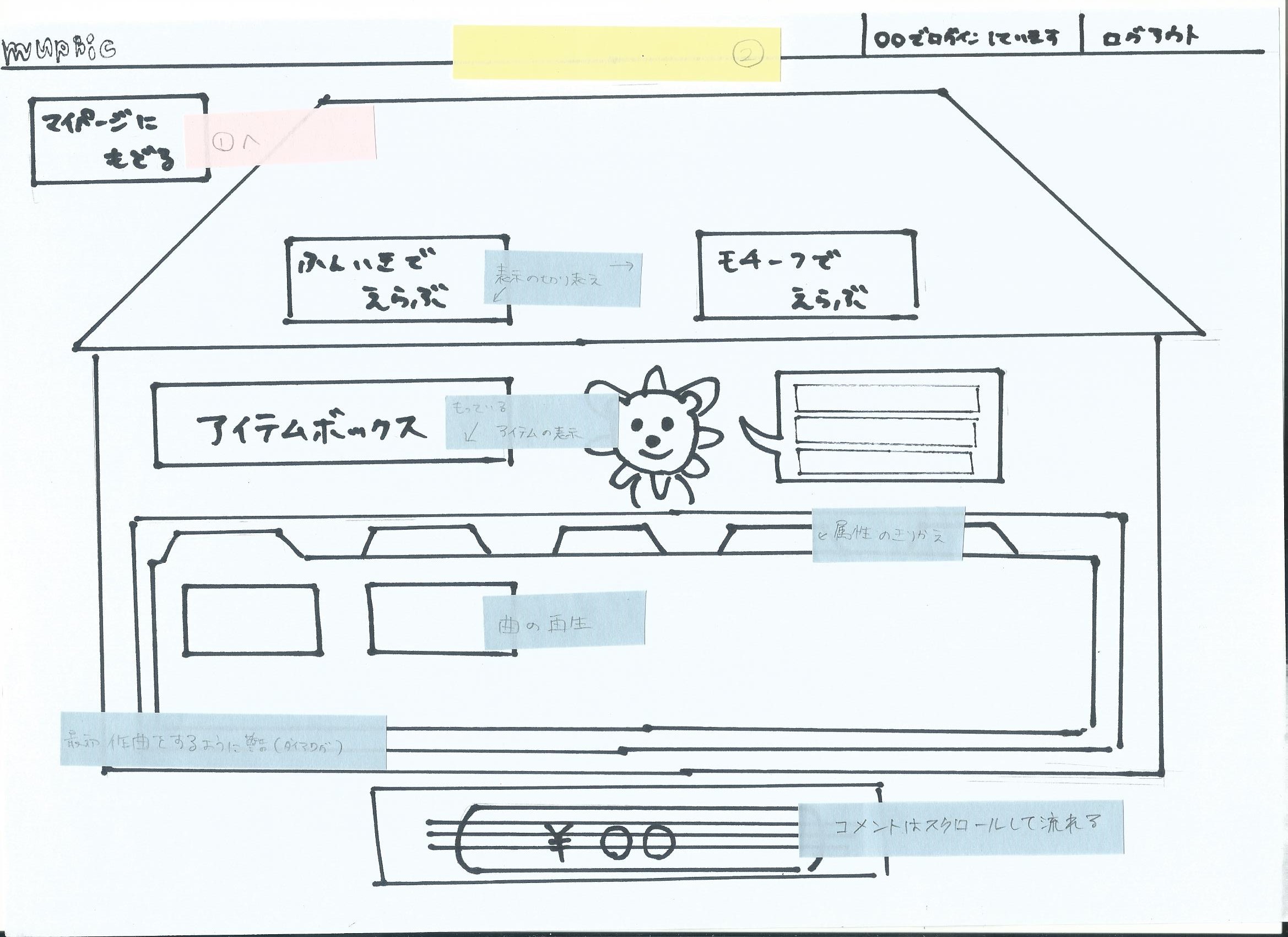
　メニュー画面は，図1.1に示す様なインターフェースである．



**図1.1** メニュー画面のプロトタイプ

　児童がWebアプリケーションにログインを行うと，メニュー画面に遷移する．メニュー画面では，児童は主に自分のお店画面への移動，友達のお店への移動，作曲画面への移動が可能である．

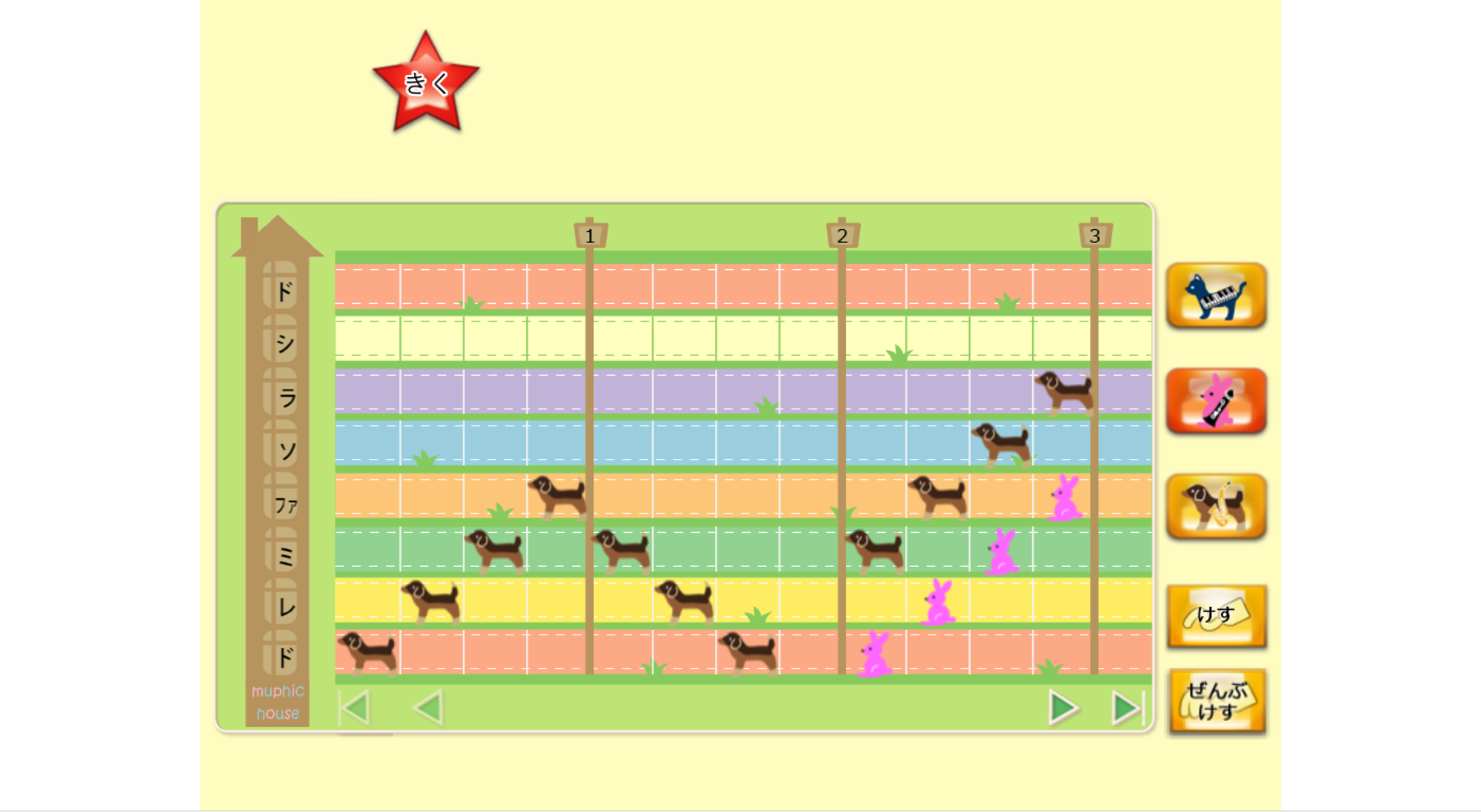
　お店画面は，図1.2に示す様なインターフェースである．



**図1.2** お店画面のプロトタイプ

　児童はそれぞれ自分のお店を持つ．新しく作曲を行う，他の児童の作品にコメントを行う事でmuphic-online内で使えるゲーム内通貨が貯まり，お店のデザインをカスタマイズする事が出来る．他の児童に対し，作成した楽曲データを売り買いする事も出来るので，それを用いてもゲーム内通貨を貯める事が出来る．

作曲は，図1.3に示す様なユーザインタフェースで行う．



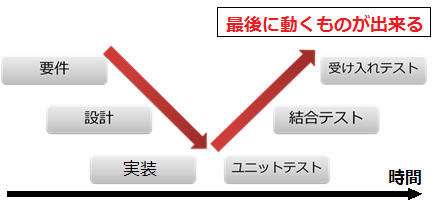
**図1.3**　作曲画面

作曲を行うにあたって，児童はピアノやオカリナ等の様々な種類の楽器を選ぶ事が可能である．楽器は動物として表現されており，画面右側に動物の絵柄で楽器のボタンが並んでいる．ボタンをクリックして楽器を選択すると，画面中央の譜面上に音符を配布する事が出来る．譜面は児童にとって難解な5線譜による表現ではなく，視覚的にも理解しやすいシリンダーオルゴールのような表現がなされている．作曲したメロディは再生する事が可能である．

## 1.3スクラム

従来ではソフトウェア開発の現場では，ソフトウェアについての要求を事前に把握し，それを分析，設計，実装し，最後に全体テストを行うという「ウォーターフォール」手法が主流であった．しかし現在，優先順位が高い機能から動くものを作り始めて短い時間で一部を完成させ，顧客に見てもらい，フィードバックを得る事でソフトウェアを成長させる「アジャイル」と総称される手法の採用が進んできている[4]．

　ウォーターフォールの開発工程は図1.4の様な手順である．

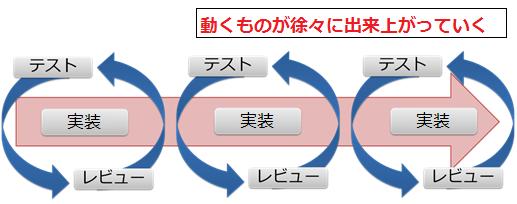


**図1.4** ウォーターフォールの開発工程

　ウォーターフォールでは，ソフトウェア全体の開発に関する工程が項目ごとに分かれている．最初に示された要件を全て完成させる事を目的とする．開発工程の流れとしては，顧客からの要件から作る必要のある物を分析し，それを元に設計を行う．その設計を元にソースコードとして機能を実装し，最後にテスト(要求や仕様を正しく満たす事が出来ているかの確認)を行う．各工程の間は仕様書を用いてやりとりする．

　問題点としては，プロジェクトの最後に，動作するソフトウェアが出来上がる為，顧客はソフトウェアが完成するまで動作を見る事が出来ない．仕様書を用いて仕様を理解する上で開発者の間で誤解が生じやすい．要求に対して完成までに時間がかかる為，ソフトウェアが完成する頃には市場の変化に乗り遅れる．といった問題が挙げられる．

　アジャイルの開発工程は図1.5の様な工程である．



**図1.5** アジャイルの開発工程

　アジャイルでは，分析，設計，実装，テストを短い期間で並列に行い，それを繰り返す．顧客からの要件全てを完成させる事が目的では無く，顧客にとって価値の高い機能から実装し，短い期間で動作するソフトウェアを完成させる[5]．短い期間で動作するソフトウェアが出来るので，顧客はソフトウェアの動作を実際に見る事が出来るので，意見を入れる事も可能となる．市場の変化に対して実装する機能を変更する，新たな要件を受け入れる柔軟性も兼ね備えている．

　muphic-online開発では，アジャイル開発手法の中でも最も普及した具体的方法の1つであるスクラムを採用している．

　スクラムでは，分析，設計，実装，テストを繰り返す短い期間の事を「スプリント」と呼ぶ．プロジェクトの内容によって期間は異なり，1～4週間に設定される．muphic-onlineでは2週間となっている．

　基本的に1回のスプリントで開発する分量を設計から取り出し，実装を行う．繰り返しスプリントを行う事で，製品をいつでも動作する状態を保つ事が出来る．

# 2研究概要

## 2.1研究の目的

　muphic-onlineはスクラムを用いて複数人で開発を行っている．しかしながら，現在の開発に置いて問題点が浮上した．

muphic-onlineはWebアプリケーションであるという特性上，実際にmuphic-onlineが正しく動作するか確認を行う場合はローカル環境では無く，実際にmuphic-onlineが稼働する環境(ネットワークサーバ)での確認が必要である．開発者は動作確認時に毎回最新のソースコードを管理システムから取得し，テスト稼働サーバへ毎回ソースコードを上げて確認を行い，更に本稼働サーバに上げるという工程を毎回行う必要がある．その解決策としてCIの導入が提案された．

　本研究では．上記問題を解決する為，また，他の簡略化可能な工程を模索する為，実際にCI環境を導入する事でその実用性について検討する．本研究で開発するCI環境をAuto Deploy System for Muphic-online(ADSM)と呼ぶ事にする．

## 2.2継続的インテグレーション

　継続的インテグレーション(以下CI)とは，ソフトウェアの開発を支援する技術の1つである．CIは，アジャイル開発を行う為の一種の実践方法として提唱されたものである．

ここで言う「インテグレーション」とは，具体的には以下の工程を指す．[6]

* 全てのソースコードを1箇所に集める
* 開発しているプログラムに依存するライブラリを包括する
* 必要な場合はビルドを行う
* データベースの構築，データのロードを行う
* 必要な場合，ミドルウェア(OS，ソフトウェア間の情報の受け渡しを行う)の設定や起動を行う．
* 単体テストと結合テスト等の予め定められたテストを実施する．

上記のプロセスを継続的に，ソースコードが更新される毎に自動的に行う．製品をいつでも動作できる状態に保つ手助けを行う．

　CIの利点としては不具合の早期発見が挙げられる．例えばソースコードを更新した時点で不具合を発見する事が出来れば，不具合を修正する事は容易であるが，ソースコードを更新してから不具合の発見までの時間が長くなる程，該当する不具合を修正する事は困難である．しかし，ビルドやテストがソースコードを更新した時点で自動的に行われるならば，不具合を早期発見し，修正する事が出来る．[6]

## 2.3研究概要

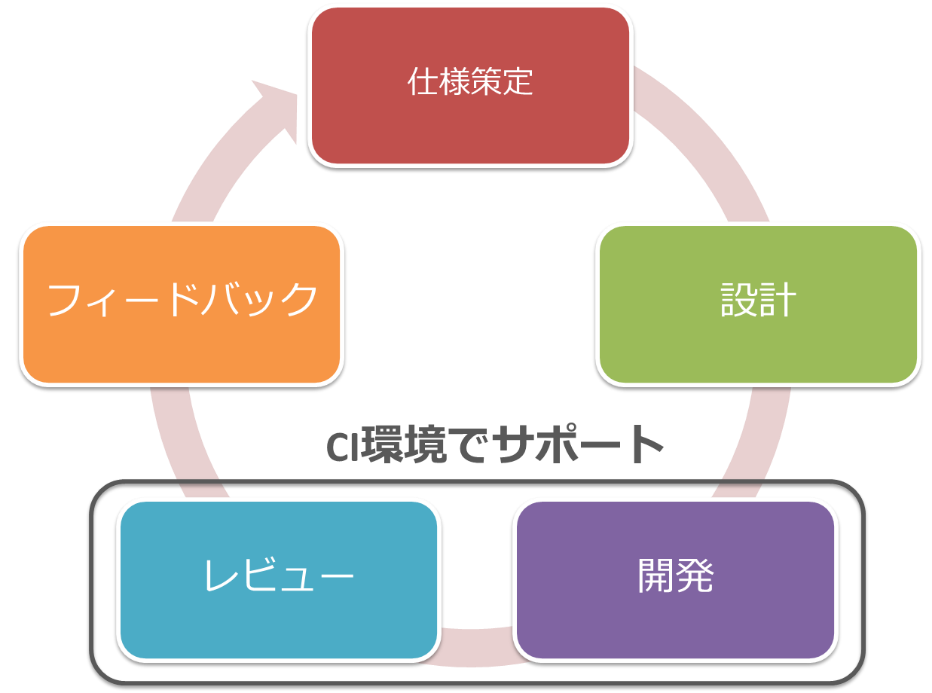
　初めに，muphic-onlineの開発工程からCI環境が適応できる部分について検討し，開発するシステムに必要な要件を導出する．それらを実現する為の要素技術(機能を成立させるための技術)を調査する．使用する要素技術が定まったら，実装を行い，システムが仕様を満たせているかを考察する．

# 3システムの要件導出

　本章では，システムの開発に当たり最初に行った要件導出についてまとめる．

## 3.1開発工程の分析

　開発工程に基づいたCI環境を開発する為，まずはmuphic-onlineの開発工程をよく知る必要がある．muphic-onlineの1スプリントの開発工程を分割すると， 図3.1の様な5つの工程に分ける事が出来る．



**図3.1** muphic-onlineの1スプリントの開発工程

* + 仕様策定

実際に使用者がソフトウェアを使う場合に必要となる機能を，使用者や顧客の要求から導出する．

* + 設計

仕様策定で決まった仕様から，その仕様の実現方法を開発者全員で考え，それぞれの仕様のアルゴリズム，プログラムの設計等の具体的な実現方法を決定する．

* + 実装

前工程までで決まった設計から，実際に開発を行う工程に入る．

* + レビュー

期間の最後に，プロダクトオーナー(製品の責任を持つ者)を交えて，この期間内に出来た部分が仕様策定で出した仕様を満たしているかどうかについてのレビューを行う．

* + フィードバック

レビュー後に，この期間内での良かった事，悪かった事、次の期間で行いたい事を開発者全員で導出し，次の期間でやる事の優先度を付ける．

　上記5工程を反復し，開発を行う．この内CI環境を用いた補助が行える工程は実装部分とレビュー部分である．他の3つの工程については，開発者自身やチーム全員で考える必要がある為，CI環境での補助は不可能である．

以下に実装部分とレビュー部分から導出された要件を示す．

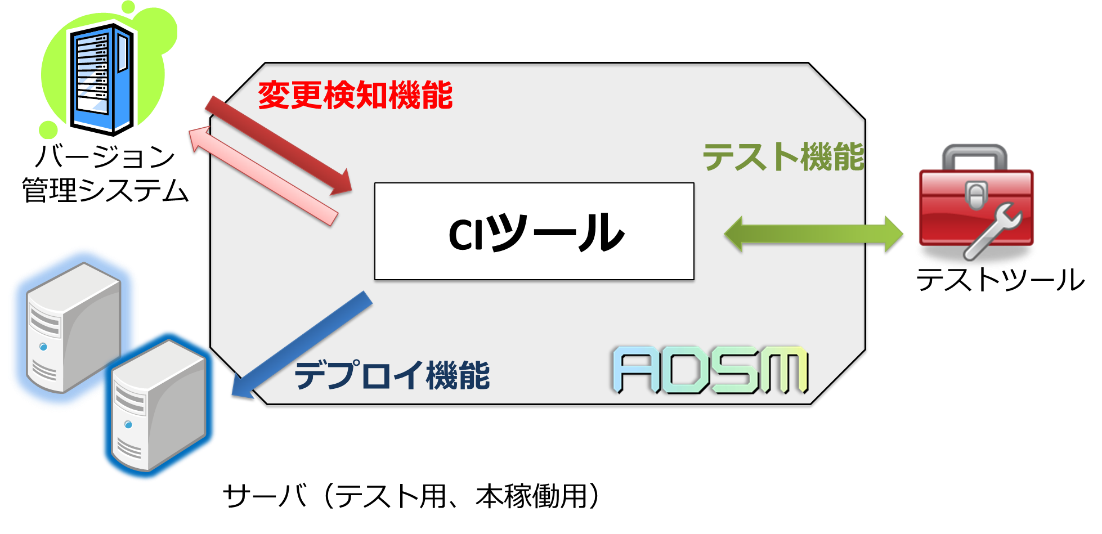
1. ソースコード管理ツールで管理されている対象プロジェクトを監視し，変更を感知出来る．
2. 対象プロジェクトの変更を感知した時，対象プロジェクトのソースコードを取得する．
3. CIツール内のプロセスを，「コードを取得→テストに掛ける→ビルドに成功すればデプロイする」という機能で分割して実装する．
4. 各プロセスを通り，異常が起こらなければデプロイを行う．

# 4システムの仕様

本章では，前章の要求分析の結果を基に定めたシステムの仕様についてまとめる．

## 4.1システムの構成

　システムの仕様を決めるにあたり，システムの全体構成を図4.1の通り3つに分割した．



**図4.1** ADSMの構成図

muphic-onlineが上げられているバージョン管理システムの監視とソースコードの取得を行う監視システムと，テストケースを実行するテスト機能，サーバへデプロイ(公開される形でアップロード)するデプロイ機能とで分けて，それぞれの仕様を定めた．

## 4.2監視システムの仕様

　監視システムは主にソースコードの変更の検知とソースコード取得の機能で構成される．ここでは，監視システムのそれぞれの仕様についてまとめる．

### 4.2.1変更検知機能の仕様

　ADSMでは、muphic-onlineが管理されているソースコード管理システムに変更があった場合，変更があった事を知る機能を実現する．その為には，ソースコード管理システムに変更があった場合，ソースコード管理システムがADSMに対し通知を送る必要がある．偽装された通知では無いかを確かめる為，ソースコード管理システムから正しい通知が送られているかをチェックする必要もある．

　他に，ソースコード管理システムには，ブランチと呼ばれる，各機能を並列して開発する為のシステムがあるが，どのブランチに対し変更が検知されたかを検知する．

### 4.2.2ソースコード取得機能の仕様

　変更検知機能によって変更が検知できた場合，ソースコード取得機能に移る．ここでは，本稼働サーバもしくはテストサーバへデプロイするソースコードを取得する．

前項で説明したブランチの内，masterブランチ(リリース可能なソースコードを含むブランチ)の変更が検知された場合，masterブランチのソースコードを取得する様にし，それ以外のブランチの変更が検知された場合はそちらのソースコードを取得する．

## 4.3テスト機能の仕様

　テスト機能では，監視システムのソースコード取得機能で得られたソースコード一式を，テストコードないしテストケース等のあらかじめ決めたテストにかける．不具合が見つからなかった場合は次のデプロイ機能へと移行する，不具合が見つかった場合は通知を行い．不具合のあるソースコードをデプロイさせない様にする．

## 4.4デプロイ機能の仕様

　デプロイ機能では，前項までの課程全てを完了したソースコードを本稼働サーバもしくはテストサーバへデプロイする．監視システムの変更検知機能で変更が検知されたブランチによって，デプロイ対象が本稼働サーバないしテストサーバに変化する．

# 5システムの要素技術

　本章では，4章で示した仕様の実装手段として採用した要素技術についてまとめる．

## 5.1 監視システムの要素技術

　監視システムの実装手段として採用した要素技術についてまとめる．

### 5.1.1 監視対象

　監視システムの要素技術として，GitHub[7]を採用した．GitHubはソースコードのバージョン管理システムをWeb上で共有できる様にしたもので，複数人で1つのプロジェクトを共有する事が出来る．muphic-onlineの開発でも使われている．GitHubは後述するCIツールに対し，ソースコードに変更が加えられたときに通知を行う機能が存在し，監視システムの通知機能の一部を実装する事が出来る．

### 5.1.2 CIツール

　監視システム，デプロイ機能の要素技術として，Jenkins[8]を採用した．CIツールはCI環境の中でも重要な存在で，設定した合図を受け取って，様々なスクリプトを動かす事が出来る．また，定期的にスクリプトを実行する事も出来る．ADSMの機能の内，監視システムの通知を読み込む，対象のソースコードを取得する，シェルスクリプト記述によるテストの実行，サーバへのデプロイを行うといった役目を担い，各システム間を連携させる．言わばコンピュータのCPU部分の様な立ち位置である．

　以下の6点が優れていると言われる[9]．

* + インストールが簡単

公式サイトからjenkins.warをダウンロードすると，起動するだけで動く．

サーバへインストールして起動する事も可能である．

* + Webブラウザだけで操作できる

基本的な設定は，Webブラウザ上で可能である．入力した設定内容はリアルタイムでエラーチェックされる．

* + 大規模でも複数環境でも対応できる

複数マシンでの分散ビルドにも対応しており，分散ビルドによって大規模なソフトウェアのビルドを行う事が出来る．

* + プラグインで好みの機能を追加できる

サードパーティのプラグインにより，好みの機能を追加する事が出来る．

世界で最も使われていると言われているだけあり様々な機能が作られており，あらゆる開発環境に対応する事が可能である．

* + オープンソースで公開されている

MITライセンスで公開されているオープンソースソフトウェアなので，誰でも無料で利用する事が出来る．ユーザコミュニティも非常に活発であり，日本ではJenkins日本ユーザ会によるJenkins勉強会が定期的に開かれる程である．

* + 様々な言語で利用できる

JenkinsはJavaで実装されている． Java以外の開発でも利用する事が出来る．例としてPythonのソフトウェア開発で利用されている．muphic-onlineもJavaでは無くJavaScriptだが，利用する事は可能である．

### 5.1.3 CIツール稼働サーバ

　CIツールを稼働させるサーバとして，Amazon Web Services[10](以下AWS)のEC2クラウドサーバを採用した．クラウドサーバなので要求される性能に合わせて処理能力を手早く変更する事が出来る．設定が終わればすぐに起動出来る事が可能であり，要求スペックが上昇した場合でも素早く能力を拡張する事が出来る．CIツールを動作させる為にはCIツールを動かすサーバが必要なので，クラウドサービスで手早く環境を整えられるAWSを選んだ．

　使用できるクラウドサーバには様々な種類があり，性能もそれによって大幅に変わるが料金が高くなる．今回は最も低い性能ではあるものの，料金は最もかからないサーバを使用する．

## 5.2 テスト機能の要素技術

テスト機能の実装手段として採用した要素技術についてまとめる．

### 5.2.1テストフレームワーク

　単体テスト機能を実現する為，JsTestDriver[11]を採用した．JsTestDriverは，Googleが作ったテストフレームワークである．Javaで書かれている為，導入も使い方もとても簡単なフレームワークで，ブラウザ上でのテストの自動化を手助けする事が出来る．実行ファイルをサーバで実行するだけでテストを行う準備が完了する為，とても扱いやすい．

　また，CIツールとの相性が良く，JsTestDriverの実行をCIツール側から自動的に行う事が出来る．

　他の利点としてはCUIからテストを実行する事が出来る事や，他のテストフレームワークからプラグインを使って移行できる事である．

## 5.3 デプロイ機能の要素技術

デプロイ機能の実装手段として採用した要素技術についてまとめる．

### 5.3.1 PaaS (Platform as a Service)

　muphic-onlineの本稼働サーバ，テストサーバを実現するため，Heroku[12]を採用した．HerokuはPaaS (Platform as a Service，ソフトウェアを稼働させるプラットフォームをインターネット経由のサービスとして提供するWebサービス)の一種であり，Webアプリケーションをデプロイする事が出来る．デプロイしたWebアプリケーションには，それぞれのアプリケーション毎に指定されるURLを用いてアクセスする事が出来る．

　HerokuはGitと呼ばれるバージョン管理システムを利用して容易にデプロイを行う事が出来る．

# 6システムの実装

　本章では，要素技術を用いたシステムの実装についてまとめる．

## 6.1 システム実装の準備

　仮想サーバ，CIツールの実装方法についてまとめる．

* CIツールを稼働させるサーバの準備

　ADSMを形作る為に，CI環境の中核であるCIツールを稼働させる仮想サーバを用意する必要があった．仮想サーバにはAWSのEC2クラウドサーバを用いた．CIツールを動作させ，外部から見る事が出来る様にするために特定のポートを解放した．仮想サーバの性能は出来るだけ高いものを選びたいが，今回は初めての環境構築の為，最も性能が低く価格が安い仮想サーバを使用した．

仮想サーバの作成は，以下の表6の様な設定にした。

**表6** CIサーバの仕様

|  |  |
| --- | --- |
| 設定項目 | 設定内容 |
| OS | Amazon Linux AMI 2014.03.1(64bit) |
| 仮想サーバの種類 | Micro Instance(t1.micro) |
| ストレージ | 8GB |
| サーバに対する接続の許可 | TCP通信:  8080番:jenkinsへ外部から接続する為に解放する．  4200番:テストツールへの接続を可能にする．  SSH通信:  22番:仮想サーバをCLIで操作する時にSSH認証が必要になる為，解放する． |
| サーバから他への接続の許可 | 全ての通信を許可 |
| IPアドレス | 固定する(今回は54.178.214.217) |

　仮想サーバの操作にはTeraTerm(接続を行うソフトウェア)を用い，SSH公開暗号鍵を用いてコマンドラインから操作を行った．

* CIツールの導入

　仮想サーバへCIツールを導入する為，CIツールであるJenkins本体を仮想サーバにダウンロードする必要がある．しかしながら，JenkinsはLinux標準のパッケージをダウンロードするコマンド (yum install)ではパッケージリストに加えられておらず，Jenkinsのパッケージがどこに存在するかを教えるファイルをダウンロードしてパッケージリストに加えたのち，インストールを行った．

　以下のリスト6.1に導入に関するコマンドと説明を記す．

**リスト6.1** Jenkinsインストール手順

|  |
| --- |
| $sudo wget –O  /etc/yum.repos.d/jenkins.repo http://pkg.jenkins-ci.org/redhat/jenkins.repo  (Jenkinsのパッケージの場所を記したファイルをダウンロードする)  $sudo rpm --import http://pkg.jenkins-ci.org/redhat/jenkins-ci.org.key  (Jenkinsパッケージの場所をインポートする)  $sudo yum install jenkins  (Jenkinsをインストールする) |

　次にJenkins起動だが，Jenkinsが常時変更を検知する等を可能にするには，Jenkinsを常時起動しておく必要がある．Linuxコマンドにはchkconfigコマンドという，サービスを自動で起動するように設定するコマンドがあるので用いた．

　リスト6.2に起動と常駐化のコマンドと説明を示す．

**リスト6.2** Jenkins導入と常駐化

|  |
| --- |
| $sudo /etc/init.d/jenkins start  (Jenkinsの起動)  $sudo chkconfig jenkins on  (jenkinsの常駐化) |

　ここまでの手順を行い，「http://54.178.214.217:8080/」にWebブラウザを用いてアクセスする事で，Jenkinsのダッシュボード画面に入る事が出来た．

* Jenkinsのセキュリティ設定

　ここまでの設定だけを行っても，Webブラウザで仮想サーバへアクセスする事で，誰でもJenkinsを操作する事が出来てしまう．また，悪意のあるユーザによって，設定を変更される事でアクセスが出来なくなる可能性や，Jenkinsのジョブ(Jenkinsに実行させる一連の流れをまとめたもの)を勝手に作られる危険性もある．これらはセキュリティ的に非常に危険なので，Jenkinsのセキュリティを設定した．

　Jenkinsの管理より設定を行い，ユーザを作成した上で，Jenkinsの各操作を行う為の権限をログイン済みユーザのみに許可する事で，匿名ユーザによる操作を不可能とした．

## 6.2監視システムの実装

　監視システムの実装では，変更検知機能とソースコード取得機能の実装を行った．各機能の概要と，実装方法についてまとめる．

### 6.2.1変更検知機能

　変更検知機能では，オンラインソースコード共有システムGitHubとJenkinsの間で通信を可能にした上で，ジョブ(一連の流れ)を作成し監視する様にした．

　実際に作ったジョブは4種類である(図6.1)．この内，名前にdeployが付いていない2つが監視システムに用いられているジョブである．



**図6.1** Jenkins上で作成したジョブ

　各ジョブは以下の仕様を持つ．

* + **muphic-online**

muphic-onlineのmasterブランチのみの変更を検知し，ソースコードを取得する．

* + **muphic-online-deploy**

muphic-onlineジョブが成功すると実行される．本稼働サーバへのデプロイを行う．

* + **TEST-muphic-online**

muphic-onlineの全てのブランチの変更を検知し，ソースコードを取得する．

* + **TEST-muphic-online-deploy**

　TEST-muphic-onlineジョブが成功すると実行される．テストサーバの内容をリセットした上でデプロイを行う．

　TESTが付いているものとついていないものの主な違いは，監視対象のブランチが異なる事である．TESTが付いている方は全てのブランチが監視対象となるテストサーバ用のジョブである．

　ここからは変更検知機能の為に行った実装方法を説明する．

* GitHubとJenkins間の連携

セキュリティ設定までの工程でJenkins自体の設定は終わったので，ここからはCI環境を構成する為に各環境を連携させる．

両者間で疎通を行う準備として，Git PluginをJenkinsに導入した．Git Pluginにより，Gitプロジェクトに関わるジョブをJenkinsで作る事を可能にする．次に，ADSM側がGitHubに対して接続を行う為には暗号鍵を作成し，それをGitHubで登録しておく必要があるので，Jenkinsユーザとして暗号鍵を作成した．

　リスト6.3に暗号鍵の作成手順のコマンドと説明を示す．

**リスト6.3** Jenkins暗号鍵作成手順

|  |
| --- |
| $cd /ver/lib/jenkins  (Jenkinsのホームディレクトリへ移る)  $sudo -u jenkins mkdir .ssh  (暗号鍵が入るディレクトリをJenkinsユーザとして作成する)  $cd .ssh  $ sudo -u jenkins -H ssh-keygen -t rsa -C jenkins@hogehoge.com  (ディレクトリ内でJenkinsユーザとして暗号鍵を作成する)  $chmod 744 ../.ssh  (セキュリティ保持の為，ディレクトリの権限を書き換える) |

暗号鍵の作成に成功したら，GitHubのユーザ設定からSSH keyを選び，暗号鍵(id\_rsa.pub)の内容をそのまま入力して保存する．保存が完了したら，仮想サーバからGitHubへ接続できるかを確かめる必要があるので，リスト6.4のコマンドを実行した．

**リスト6.4** GitHubに対する接続確認

|  |
| --- |
| $sudo –u jenkins ssh –T git@github.com |

アクセスに成功すると，認証に成功したという文面の英語の文章が表示される．最後に，リスト6.5のコマンドを実行し，JenkinsのGitとしてのユーザの登録を行う．

**リスト6.5** Gitユーザ登録

|  |
| --- |
| sudo -u jenkins git config --global user.email "jenkins@hogehoge.com"  sudo -u jenkins git config --global user.name "jenkins" |

### 6.2.2ソースコード取得機能

　ソースコード取得機能では，変更検知機能で示したジョブでソースコードを取得する様にした．Jenkinsのジョブの中には，ビルド・トリガと言う機能があり，この機能の中にある，バージョン管理システムを監視する機能をトリガとして，ソースコードを取得する様にした．

　また，取得したソースコードはワークスペースと呼ばれる，Jenkinsが動いている仮想サーバ内に確保された領域に対し保存される．しかし，標準ではログインを行わなくともワークスペース内を観覧する事が出来てしまうので，ソースコードを取得してからデプロイ機能を行うまでソースコードを保持し，全ての作業が終わったらワークスペース内を削除する必要があった．そこでWorkspace Cleanup PluginをJenkinsにインストールし，ジョブが終了する時に実行する事で問題を解消した．

ここからはソースコード取得機能の為に行った実装方法を説明する．

* GitHubの更新時，Jenkinsで取得する
  + GitHub側

　対象プロジェクトの設定からWebhook&Servicesを選択．Add ServiceからAdd Jenkins(Git plugin)を選択する．するとJenkins URLを入力する欄が出現するので，URLを入力する．(http:// 54.178.214.217:8080/)

* + Jenkins側

新規ジョブ作成からGitを選択し，URL欄に対象リポジトリのURLを入力する．(git@github.com:[ユーザ名]/[プロジェクト名].git)

「SCMをポーリング」にチェックを入れる．(対象プロジェクトの変化を察知するにはこれにチェックを入れる必要がある．)

## 6.3テスト機能の実装

　テスト機能の実装では，テストフレームワークの導入を行った．ここでは機能の概要と，実装方法についてまとめる．

　テストフレームワークを動作させるサーバと，Jenkinsが動作するサーバは同じ仮想サーバとする．今回使用するJsTestDriverを導入するため，googleの公式配布サイトより，最新(2015/02/10現在 ver1.3.5)のJsTestDriverをダウンロードする必要がある．

　JsTestDriverのテストの流れは以下の流れで行う．

1. テストを行う時に，仮想サーバにJsTestDriverを実行させウェブブラウザ上から観覧できるようにする(リスト6.6)．Jenkinsと同じポート8080を使わない様に注意する．

**リスト6.6** JsTestDriver起動

|  |
| --- |
| $java –jar JsTestDriver-1.3.5.jar –port4200 |

1. JsTestDriverはブラウザを用いてテストを行うので，テスト前にブラウザでアクセスする必要がある．仮想サーバのPort4200(http://54.178.214.217:4200/)にアクセスし，Capture This Browserを選択して待つ
2. リスト6.7の様にコマンドを打ち込み，テストを実行する．

**リスト6.7** テスト実行コマンド

|  |
| --- |
| $java -jar JsTestDriver-1.3.5.jar --tests all |

## 6.4デプロイ機能の実装

デプロイ機能の実装では，テストサーバと本稼働サーバへのデプロイを前工程から自動的に行う機能を実装した．ここでは機能の概要と，実装方法についてまとめる．

　デプロイ機能は，図6.1のジョブの内，名前にdeployが付いているジョブが担当している．ジョブを2つに分け，それぞれ本稼働サーバとテストサーバに対しデプロイを行う様にした．また，テストサーバに対してのデプロイについては，全く異なるブランチの内容がデプロイされるとGitの性質上，変更が拒絶される点を踏まえて，デプロイの前に毎回テストサーバを初期状態にリセットする事で回避する様にした．

ここからはソースコード取得機能の為に行った実装方法を説明する．

* Herokuへの自動デプロイ

　Herokuに対する接続も，GitHubと同様に暗号鍵を用いた接続方法が主となる．GitHubの接続時に作成した暗号鍵(id\_rsa.pub)の中身をHerokuの設定画面の鍵登録画面に入力して保存する．保存後，ADSM仮想サーバ側から接続確認を行う必要があるので，リスト6.8のコマンドを実行した．

**リスト6.8** Herokuに対する接続確認

|  |
| --- |
| $sudo –u jenkins ssh –T git@heroku.com |

　接続が確認できたら，Herokuでテストサーバ，本稼働サーバに当たるアプリケーションを作成しておく．

* Jenkinsのジョブを設定

　ここでは図6.1で挙げたジョブの内，deployが付いている方について説明する．Jenkinsの仕様により，ソースコードを取得すると、どのブランチからソースコードを取得したとしてもワークスペース内のソースコードはmasterではないブランチに存在するとして扱われる．これはmasterブランチを守る策の1つであるが，Herokuへデプロイを行う場合，Herokuのmasterブランチへデプロイを行う必要がある為，取得したブランチをmasterブランチとして扱う設定を行う．

JenkinsからHerokuへデプロイする為に，ビルド手順の追加からシェルスクリプトを選択する．リスト6.9にシェルスクリプトの内容を示す．Herokuアプリのリモートリポジトリを知らない場合は教えた上でデプロイを実行し，既に場所が分かっている場合は普通にデプロイを行うといった内容である．

**リスト6.9** デプロイシェルスクリプト

|  |
| --- |
| if ! git ls-remote heroku; then  git remote add heroku git@heroku.com:[herokuアプリ名].git  fi  git push heroku master |

# 7システムの考察

　本章では，実装したシステムの監視システム，テスト機能，デプロイ機能の考察についてまとめ，全体としてユーザの要求を満たすシステムが開発できたかを考察する．

## 7.1監視システムの考察

　監視システムの仕様の充足と，実際の工夫点についての考察をまとめる．

### 7.1.1監視システムにおける仕様の充足について

　実装したシステムが監視システムの仕様を満たせているか考察する．

　変更検知機能に関しては，開発者が気にする事無くデプロイが行われた時点でプロジェクトの更新を検知する事が出来た為，監視システムの仕様を満たす事が出来た．また，監視対象の変更は，「Jenkinsのmuphc-online(TEST\_muphic-online)ジョブの対象リポジトリURL」と，「GitHubのJenkins(Git plugin)内のCIサーバ側のURL設定」の2箇所を変更する事で柔軟に変更する事が可能である．どちらもCLIの様なコマンドの知識を必要とする仕様では無く，ブラウザからGUIで設定する事が可能であり，敷居も低い．

　ソースコード取得機能に関しては，前工程からそのまま続けて実行する様に行った．監視システムで指定されている対象リポジトリからソースコードを取得する機能を実装する事が出来た．本稼働サーバへのデプロイを行うジョブと，テストサーバへのデプロイを行うジョブによって取得する対象のリポジトリが異なる様にする事で，仕様を満たす機能を構成する事が出来た．

　各機能の仕様の達成度から，監視システム部分は開発出来たと考えられる．しかしながら，検知を行い，ソースコードが取得出来たという事実自体を開発者が知る為には，ブラウザからジョブの進行状況を確認しなければいけないという点も見られる．変更の検知もしくは取得が完了した時点で，開発者に何らかの方法で完了したという事を知らせる方が更に開発者にとって楽であると考えられる．

### 7.1.2監視システムにおける実際の工夫点について

　監視システムにおける実装の工夫点について考察する．

* 本稼働用ジョブとテスト用ジョブの分割

本稼働用の変更であれば本稼働サーバへデプロイ，それ以外テスト等での変更であればテストサーバへデプロイという動作を実現させるために，変更対象がmasterブランチかそうでないかで，取得するソースコードを変更する必要があると考えた．その為，本稼働用とテスト用の2つにジョブを分割し，必要な時にどちらかのジョブが自動的に動く事で実装を簡単かつ分かりやすくした．

* 操作が簡単

　ジョブの作成，設定にはコマンドラインを用いた操作，設定は基本的に必要無く， GitHub公式サイトやJenkinsのダッシュボード画面からジョブの設定を行う事で，プログラムやサーバ関係の知識が少ない人であっても比較的簡単にジョブを作成する事が出来，直感的に操作する事が出来る．

## 7.2テスト機能の考察

　テスト機能の仕様の充足と，実際の工夫点についての考察をまとめる．

### 7.2.1テスト機能における仕様の充足について

　実装したシステムがテスト機能の仕様を満たせているか考察する．

　今回はテストフレームワークJsTestDriverの導入と，JeTestDriverが実行されている時に特定ポートで仮想サーバをブラウザ観覧すると正しく画面が表示される事が確認でき，動作を確認する事が出来た．しかし，muphic-onlineに基づいたテストケースを準備する事が出来なかった為，muphic-onlineに対するテスト機能実施を行う事が出来なかった．その為，Jenkinsのジョブの中に組み込む事がまだできておらず，仮想サーバへ接続しコマンドラインで起動して動作を確認したのみに留まった．

　今後ADSMを運用する場合，まずmuphic-onlineに基づいたテストケースの作成が必要である．次に，muphic-online-testジョブを作成し，現存のmuphic-onlineジョブとmuphic-online-deployジョブの間に挟む構成が必要と考えられる．

　仕様の達成度から，導入のみは完了したが運用出来る状況ではない為，仕様を完全に満たす事は出来なかった．

### 7.2.2テスト機能における実際の工夫点について

　テスト機能における実装の工夫点について考察する．

* JenkinsとJsTestDriverの組み合わせ

　JenkinsとJsTestDriverはどちらもJavaで書かれているという時点でまず共通点がある．他に挙げられるこの組み合わせの良い点としては，JsTestDriverでテストを行う時の実行方法がコマンドであり，テストを開始する，テストを行うといった動作をシェルスクリプトで記述する事が出来る．Jenkinsにはビルド終了時に任意のシェルスクリプトを実行する機能があり，この2つを組み合わせる事によってテストの自動化を比較的簡単に実装する事が出来る．

## 7.3デプロイ機能の考察

　デプロイ機能の仕様の充足と，実際の工夫点についての考察をまとめる．

### 7.3.1デプロイ機能における仕様の充足について

　実装したシステムがテスト機能の仕様を満たせているか考察する．

　設計で示したデプロイ用のシェルスクリプトを用い，デプロイが正しく行われる事が確認できた．また，masterブランチ以外の変更時は全てテストサーバへデプロイが行われる事が要求として示唆されたが，実装を行い，正しく動作する事が確認できた．

しかしながら，デプロイしたアプリケーションがアプリケーションエラー扱いになる現象が確認された．エラーが起きる条件は不明だが，対象のアプリケーションのDynos(仮想環境のスペック値)が何らかの理由で0に設定され，起こっているという事が判明した．対処方法としては，herokuの対象アプリケーションを，heroku restartコマンドで再起動する，heroku公式サイトにおいて，エラーが起きているアプリケーションのDynosを0から1に変更するという2通りの方法が挙げられる．

　仕様の達成度から，アプリケーションエラーはheroku側の問題とすると，デプロイ機能の仕様は達成できたと考えられる．

### 7.3.2デプロイ機能における実際の工夫点について

　デプロイ機能における実装の工夫点について考察する．

* テストサーバへのデプロイ

　テストサーバへのデプロイは，masterブランチ以外の全てのブランチが対象となる．しかし，Gitのバージョン管理システムの仕様により，連続的ではない変更はrejected(除外)されてしまう．しかし，デプロイ機能としては，その時変更があったブランチからソースコードを取得し，毎度同じテストサーバにデプロイしたい．

　前述の問題を解決する為，「Heroku Plugin for Jenkins」というプラグインをJenkinsに導入した．このプラグインはJenkinsからHerokuへのデプロイを手助けするプラグインで，普段コマンドラインから打たなければいけない操作を自動で行ってくれる．今回はこのプラグインの機能の中からHeroku:Rollbackという機能をデプロイ前に使う事にした．Heroku:Rollbackを使うと，Heroku上のアプリケーションを1つ前のバージョンに戻す事が出来る．つまり，アプリケーションをデプロイする前に毎回1つ前のバージョンに戻され，何もデプロイされていない初期バージョンへ初期化される様になる為，同じテストサーバで複数のブランチを扱う事が出来る様になる．

## 7.4 システム全体における要求の充足について

　実装したシステムが開発者の要求を満たせているか考察する．「開発者の負担の改善」，「開発工程の簡略化」，「本稼働，テストサーバの活用」の3つの観点から，要求の充填に関する考察を述べる．

　初めに示された問題である，ネットワークサーバへのデプロイが毎回必要になってしまっていた問題については，開発者の負担を改善したいという要求があった．この要求に対しては監視システムによって常に変更を検知し，変更をデプロイ機能でデプロイするという2つの機能が実装されたことにより，開発者の負担を減らすという要求を満たす事が出来たと考えられる．

　開発工程の簡略化について上記の2つの機能が実装された為，開発者がコマンドラインを開き無駄なコマンドを打つ必要がなくなった点から考えれば要求を満たせているかもしれないが，テスト機能の部分を実際のmuphic-onlineに当てはめて実装する事が出来なかった事から，工程の簡略化について全ての要求を満たす事は出来なかったと考えられる．

　本稼働，テストサーバの活用については，デプロイ機能でテストサーバに対してあらゆるブランチからのデプロイが1つのテストサーバのみで可能になった為，前年度よりは活用できていると考えられる．本稼働サーバに対するデプロイも自動で行われる事も踏まえると，本稼働，テストサーバの活用に関しては要求を実現できていると考えられる．

　以上の考察から，ADSMシステム全体として，開発者視点からの要求を概ね満たす事が出来たと考えられる．

# 8今後の展望

　本章では，開発したシステムに関して今後実践すべきことについてまとめる．

## 8.1システムの評価

　本研究で開発したADSMはシステム全体として要求と仕様を概ね満たしているが，システムが開発に対してどの程度価値があるのかを評価出来ていない．したがって，本研究で開発したADSMを使用する場合としない場合の開発速度についての検証を行う必要があると考えられる．

## 8.2テストケースの導入

　現在muphic-onlineにはテストケースが存在しておらず，今回ADSMではテストフレームワークの導入は行ったものの，muphic-onlineのテストケースについては作成されていない．その為テストケースを作成し，単体テストやユニットテストを行う必要があると考えられる．

　本研究中でも不具合が発生し，原因の解明が非常に遅れた事があった．ローカル環境では正常にmuphic-onlineは起動する事が確認できたはずが，テストサーバへデプロイを行うと何故か動作しないというものだった．原因はファイル名のミスであった．全てのファイル名の頭文字は大文字で命名されているのだが，ある1つのファイル名の頭文字が小文字だった為である．しかし，windows上では大文字と小文字を同じ文字として扱う為，動作しており，テストサーバはlinux系統で大文字と小文字を区別する為，動作しないとなる．

　上記の不具合を発見し，修正する為に2週間以上の時間をかけてしまった．特にローカル環境で正常に動作するという点があった為でもある．テストケースが導入される事で，不具合の早期発見が期待できる．

# 9おわりに

　本研究では，muphic-onlineの開発工程に基づいたCI環境として，「Auto Deploy System for Muphic-online(ADSM)」を開発した．最初にmuphic-onlineの開発工程からシステムの要求分析を行い，システムの仕様を定めた．そしてそれらを実現する為に必要な要素技術を用いて実装を行った．最後に，開発したシステムについて要求に対する満足度を考察し，今後の展望をまとめた．その結果，開発者に手間をかけさせる必要が無く自動でデプロイを行うCI環境を開発する事が出来た．

　最後にシステムの運用に関して結論をまとめる．本研究で開発したADSMは要求と仕様をほぼ満たしていると考察出来たが，現状ではまだ十分な運用は困難であると結論付ける．その理由としては，テストケースが存在しない為，テスト機能を動かす事が出来ない為である．テストサーバを用意してあるとはいえ，テストサーバへアクセスし，全ての不具合を毎度チェックしていたのでは時間がかかってしまう．勿論テスト機能やテストケースだけでは不具合を完全に防ぐ事は出来ないが，全ての作業を人間が行うよりは効率的だろう．よって，テスト機能を工程の中に組み込み，開発者の作業を効率化していくようなシステムとしてADSMを開発していく必要がある．

# 共同研究者の紹介

　本研究を行うにあたって，1年間共にmuphic-onlineの開発に取り組んだ力武研究室所属の共同研究者を紹介しておく．

* 情報電子システム工学専攻科 2年　高橋綾（敬称略）
* 情報電子システム工学専攻科 1年　山田希望（敬称略）
* 情報システム工学科 5年　三森理司

# 謝辞

　本研究を進めるに当たり，研究や卒業論文執筆に関してご指導を頂き，研究に必要なサーバの確保等を行って下さった同研究室の力武克彰助教授に感謝致します．また，様々な指摘やアドバイスを下さった同研究室の皆様に感謝致します．

# 参考文献

1. 亀谷学人，力武克彰，佐藤貴之「児童対象メロディ付き物語創作支援システムの開発」『情報処理学会創立50周年記念（第72回）全国大会講演論文集』4号，pp.595，2010
2. 佐藤貴之，石澤慶子 他「児童対象メロディ付き物語創作支援システムの開発」『情報科学技術フォーラム講演論文集』7号，pp.73，2008
3. 山下峻，メロディ付き物語創作支援Webアプリケーションの開発，卒業研究論文，pp.2，2014
4. 平鍋健児，野中郁次郎『アジャイル開発とスクラム:顧客・技術・経営をつなぐ協調的ソフトウェア開発マネジメント』翔泳社
5. アジャイル開発の本質 〜 アジャイルとウォーターフォールの違いとは | Social Change! <http://kuranuki.sonicgarden.jp/2013/07/アジャイル.html> (2015/02/09アクセス)
6. 池田尚史，藤倉和明，井上史彰(2014)『チーム開発実践入門:共同作業を円滑に行うツール・メソッド』技術評論社 p.147-268
7. GitHub

<https://github.com/> (2015/02/09アクセス)

1. Jenkins

<http://jenkins-ci.org/> (2015/02/09アクセス)

1. 和田貴久，河村雅人，米沢弘樹，山岸啓(2011)『Jenkins実践入門:ビルド・テスト・デプロイを自動化する技術』技術評論社
2. Amazon Web Services

< http://aws.amazon.com/jp/> (2015/02/09アクセス)

1. JsTestDriver

< http://code.google.com/p/js-test-driver/> (2015/02/09アクセス)

1. Heroku | Cloud Application Platform

<https://id.heroku.com/login> (2015/02/09アクセス)