ｇ－ｍｏｄ
プロジェクト報告

Ver1.1

g-mod
Project Manager 谷田部 学
環境情報学部４年　佐藤 俊之
環境情報学部４年　平澤 秀幸
環境情報学部３年　松田 航

2007 年 1 月 31 日作成
文書番号：gm-070123-100-My001

I. 目次

II.　プロジェクト概要

1.　クライアントニーズ

慶応大学　DegitalMediaContents（DMC）機構の活動としてDMCネットワークシステム開発を行っている.
DMC での２００６年度ミッションとして「知の再編」を掲げており，その一つとして蓄えられた知識を誰もが判り易く引き出すことが出来るような方法で提供する事を考えている.
具体的な提供方法の一つとして，地図を利用する事による「コンテンツ情報取得の簡易化」を考えている.
その他にも今までに無いような新しい提供方法があれば実現していきたいと考えている.

2.　クライアント要件

DMCにおける２００６年度ミッション「知の再編」を具現化する.
具現化のひとつとして，既に運用されている発想支援システム「DMCネットワークシステム」に新機能を追加する.
新機能とは各コンテンツが属性として持っている緯度経度情報を利用して地図上にコンテンツ目印を配置する機能である.
この新機能の核となるモジュールを開発・実装して欲しい.

3. 開発する機能

以下の機能を実現するシステムの開発を行う.

1) **システムが受け取った地名情報を元に緯度経度を返信する.**
2) **システムが受け取った地名情報を元にDBに登録されている緯度経度情報を検索する.**
3) **地名情報がDBに登録されていないとき，インターネットで公開されている緯度経度情報算出APIに問い合わせて緯度経度を返信する.**
4) **緯度経度情報算出APIに問い合わせた緯度経度情報をDBに保存する.**

4. プロジェクトの目的と目標

目的:使われるシステムを作る！

使われないシステムを作っても意味が無い.

クラインとの要望通りに作ることも大切だが，使われるようにする為の提案し，実現することもこの目的に含まれている.

重点目標:最良品質システム開発

使われるシステムを作るためにはとにかく品質が重要.

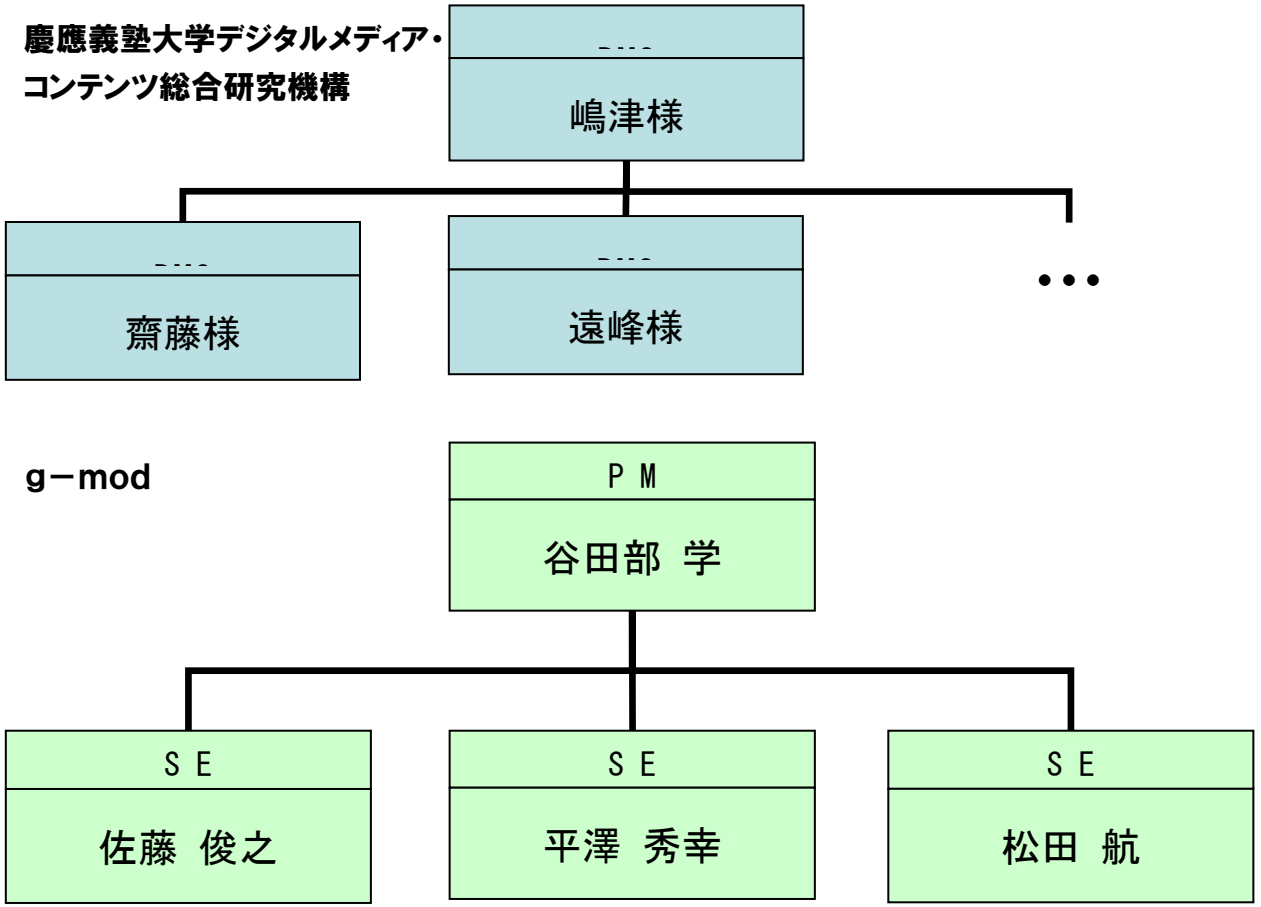
品質が低くては使いものにならない.

出来るだけ良い品質のシステムを作る為に様々な工夫と努力をする.

学習目標:新しいことへの挑戦

プロジェクトを通して，今まで行っていなかったことや苦手だったことに積極的にチャレンジして学んでいく.

5.　プロジェクト体制



6. プロジェクト期間

本プロジェクトは下記の期間で行われる.

2006 年 10 月 5 日　～　2007 年 1 月 31 日

III. プロジェクト提案

1. 作業範囲

納品物（後述）の**納品及びモジュールの初回導入を行うまで**をプロジェクトの作業範囲として定義する.
作業の担当は DMC 様に指定されたものに基づいているが，開発環境構築や保守の方法については g-mod チームの提案を元に決定した.
それ以外の項については指定された内容で構わないということで，指定通りに行うこととなった.

本プロジェクトで行う作業の担当割り振りは以下の通りとする.

No	名称	担当	
		DMC 様	g-mod
1	要求仕様	○	
2	機能仕様	○	
3	プロジェクト定義		○
4	プログラム仕様		○
5	開発環境構築	○	○
6	結合テスト環境構築		○
7	キャッシュテーブルのレイアウト		○
8	開発		○
9	テスト（単体，結合）		○
10	テスト報告（単体，結合）		○
11	ドキュメント作成		○
12	テスト結果検証		○
13	本番環境配備　(※初回のみ)		○
14	テスト環境配備		○

15	本番運用	○	
16	保守	○	

このうち，５．開発環境構築は，セキュリティの面から OS の導入及びアカウントの作成を DMC 様に，
必要なソフトウェアの導入は g-mod が担当となった．

2. 開発環境

指定されていた開発環境は座標検索 API のみであり, それ以外の開発環境はチーム内で考案し, DMC 様に提案し許可を頂いたものとなっている. チーム内で考案した内容の根拠であるが，メンバーの経験及び知識があるものを重視して選択した.

開発環境は以下のようになっている.

使用言語：JDK 1.5.10
実装環境：Eclipse 3.2.1
座標検索 API：指定された Geocoding API （http://www.geocoding.jp/api/）を用いる
外部ファイル：テキストファイル（設定ファイル，プログラムとリクエストログの形式．拡張子ではない）
データベース：MySQL 5.0.24
出力及び入力のエンコード：UTF-8

3. 成功報酬

遠峰様と話し合った結果，プロジェクトの目的が達成された暁には以下のような報酬を頂く方針でプロジェクトを進めることになった.

- 大岩研究室が，DMC システムのコンテンツアップローダー権限を貰う
- リンク一覧に大岩研究室もしくは g-mod チームのリンクを張っていただく

4. 納品物

本プロジェクトに対する納品物は以下の通りとなっている.

No.	名称
1	プロジェクト提案書
2	プログラム仕様書
3	プログラムソース
4	実行モジュール
5	単体テスト仕様書
6	単体テスト結果報告書
7	結合テスト仕様書
8	結合テスト結果報告書
9	インストール手順書
１０	稼動手順書

5. 品質基準

品質基準は，クライアントからの要求を元にチーム内でこのシステムを開発するに当たって必要最低限となる品質基準を検討した．提案した内容は以下のようになっている．

1) 規約

コーディングの規約は，電通国際情報サービス社による規約（添付：JavaCodingStandard2004.pdf 参照）を用い，
この規約にできる限り沿うこととする．

2) レビュー

下記の工程にてレビューを行なう事とする．

(1) プロジェクト提案書（g-mod 内部レビュー）
(2) 詳細設計書（g-mod 内部レビュー）
(3) テスト方式設計書（g-mod 内部レビュー）
(4) 単体テスト仕様書（DMC 様レビュー，g-mod 内部レビュー）
(5) 結合テスト仕様書（DMC 様レビュー，g-mod 内部レビュー）
(6) 単体テスト結果報告書（DMC 様レビュー，g-mod 内部レビュー）
(7) 結合テスト結果報告書（DMC 様レビュー，g-mod 内部レビュー）

3) テスト

(1) テスト仕様

単体・結合共に，テスト仕様前に DMC 様による承認を得てからテストを行うこととする

(2) レビュー

単体・結合問わず，全てのテストに置いて，テスト結果を記録する．
品質基準を達成しているかは，テスト仕様が受理された後でテストを達成することが判断基準となる

(3) テスト自動化

単体テストにおいては，可能な限りテストは，テスティングフレームワークである JUnit を用いて行うこととする．

IV. プログラム仕様

1. 用語一覧

Map 型 UI： DMC ネットワークシステムの新機能.（詳細は「DMC ネットワークシステム Map 型 UI 表示機能　機能仕様書」を参照）
GeocodingModule： Map 型 UI の一部．渡された地名情報を処理し，それに該当する緯度経度情報を返す.
緯度経度情報キャッシュテーブル： 緯度経度情報を蓄えるデータベース．以降「キャッシュテーブル」とも表記.
Geocoding API： 外部に提供されている，緯度経度算出サービス．緯度経度情報キャッシュテーブルに存在しない緯度経度情報を検索するために使用する．今回は Google 社のものを使用する（http://www.geocoding.jp/api）.
同名異地点： 「DMC ネットワークシステム Map 型 UI 表示機能　機能仕様書－用語一覧」を参照
設定ファイル： GeocodingModule の動作を制御する設定ファイル.
システムログ： GeocodingModule の動作履歴が記憶されているログファイル.
アクセスログ： GeocodingModule に渡されたリクエスト内容が記憶されているログファイル.

2. はじめに

単純なシステムでありながら，GeocodingModule は最良品質のものを目指して設計された．システムの動作履歴はすべてシステムとアクセスログに残され，ユーザが管理しやすいものになっている．重要なプログラム設定はすべて外部の設定ファイルから読み込む仕組みになっており，GeocodingModule の動作を自由に変更できる．またデータの独立性を図り，データベースのテーブル設計も単純かつ管理しやすいものになっている.

3. 概要

GeocodingModule は，DMC ネットワークシステムの新機能である Map 型 UI の一部である．渡された地名に該当する緯度経度情報を返す単純なシステムです．DMC ネットワークシステムにおける GeocodingModule の位置づけは図 1 と「DMC ネットワークシステム Map 型 UI 表示機能機能仕様書」を参照．

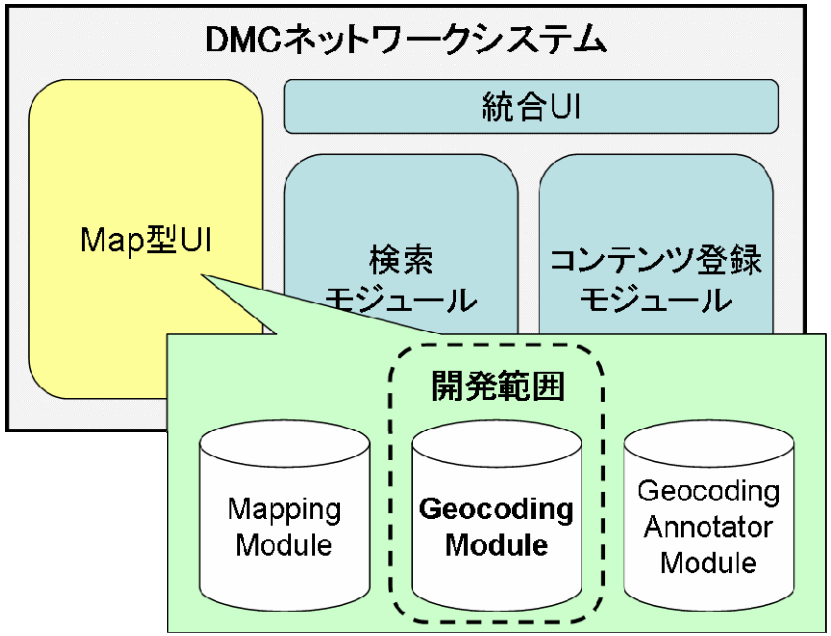
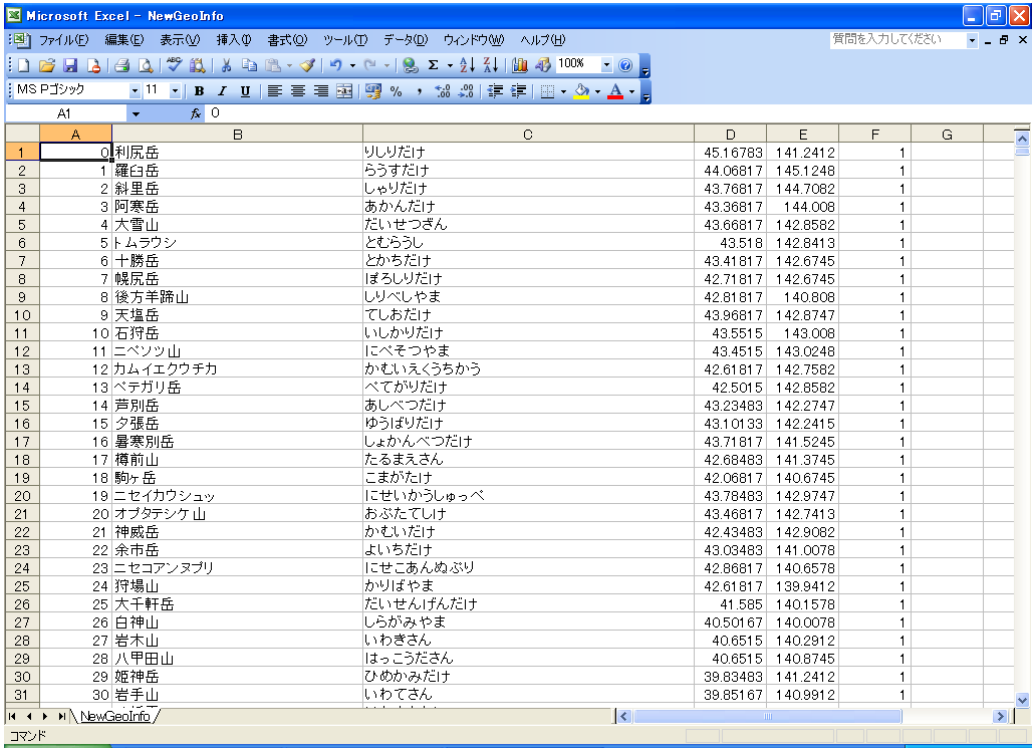


図 1

4. Module の特徴

GeocodingModule の導入前までは，マップ表示に必要な地理情報は膨大な CSV ファイル形式で管理されていた．実際に使われていたファイルは下の図のようなものである．



	A	B	C	D	E	F	G
1	0	利尻岳	りしりだけ	45.16783	141.2412	1	
2	1	羅臼岳	らうすだけ	44.06817	145.1248	1	
3	2	斜里岳	しゃりだけ	43.76817	144.7082	1	
4	3	阿寒岳	あかんだけ	43.36817	144.008	1	
5	4	大雪山	だいせつざん	43.66817	142.8582	1	
6	5	トムラウシ	とむらうし	43.518	142.8413	1	
7	6	十勝岳	とかちだけ	43.41817	142.6745	1	
8	7	幌尻岳	ぽろしりだけ	42.71817	142.6745	1	
9	8	後方羊蹄山	ししりべしやま	42.81817	140.808	1	
10	9	天塩岳	てしおだけ	43.96817	142.8747	1	
11	10	石狩岳	いしかりだけ	43.5515	143.008	1	
12	11	ニペソツ山	にぺそつやま	43.4515	143.0248	1	
13	12	カムイエクウチカ	かむいえくうちかう	42.61817	142.7582	1	
14	13	ペテガリ岳	ぺてがりだけ	42.5015	142.8582	1	
15	14	芦別岳	あしべつだけ	43.23483	142.2747	1	
16	15	夕張岳	ゆうばりだけ	43.10133	142.2415	1	
17	16	暑寒別岳	しょかんべつだけ	43.71817	141.5245	1	
18	17	樽前山	たるまえさん	42.68483	141.3745	1	
19	18	駒ヶ岳	こまがたけ	42.06817	140.6745	1	
20	19	ニセイカウシュッ	にせいかうしゅっぺ	43.78483	142.9747	1	
21	20	オプタテシケ山	おぷたてしけ	43.46817	142.7413	1	
22	21	神威岳	かむいだけ	42.43483	142.9082	1	
23	22	余市岳	よいちだけ	43.03483	141.0078	1	
24	23	ニセコアンヌプリ	にせこあんぬぷり	42.86817	140.6578	1	
25	24	狩場山	かりばやま	42.61817	139.9412	1	
26	25	大千軒岳	だいせんげんだけ	41.585	140.1578	1	
27	26	白神山	しらがみやま	40.50167	140.0078	1	
28	27	岩木山	いわきさん	40.6515	140.2912	1	
29	28	八甲田山	はっこうださん	40.6515	140.8745	1	
30	29	姫神岳	ひめかみだけ	39.83483	141.2412	1	
31	30	岩手山	いわてさん	39.85167	140.9912	1	

※　16162 行目まで同じような状態が続いている

地理情報を上のような形で管理すると，次のような問題が発生する．

- 地理情報を手作業で追加しなければならない
- 地理情報を手作業で更新しなければならない

- 上の二つの作業を行っている最中に発生する人間のミス
- 地理情報の追加日時や更新日が分かりにくい
- エントリーが重複することがある

GeocodingModule は上記の問題を解決し，今まで手作業で起こっていた作業を全て自動化してくれる．そして，次のような機能を備えている．各機能の詳しい説明は後述に記載されている．

- 地理情報の追加／更新の自動化
- 地理情報を常に最新の状態にする
- 地理情報の追加日時や更新日は登録され，自動的に管理される
- エントリーの重複を防ぐ
- 地名改正などの例外事態への対応性
- 使用環境に合わせて，設定ファイルで GeocodingModule の動作を制御に変更できる
- GeocodingModule の動作履歴がシステムログとアクセスログという形で残る

5. Module 構造

GeocodingModule は緯度経度情報キャッシュテーブルと設定ファイルの二つのコンポーネントから構成される．下の図 2 に GeocodingModule と各コンポーネントとの関連性を示す．

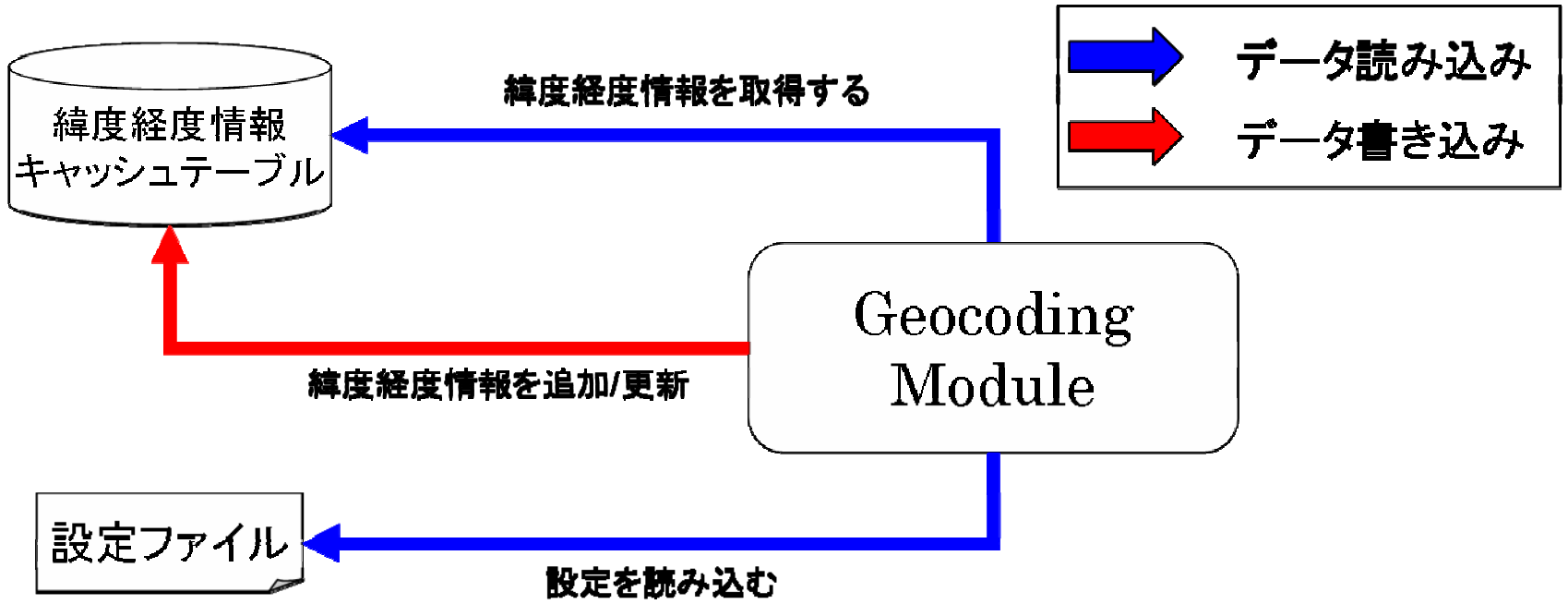


図 2

6. 入出力

Gecoding Module はソケット通信を通してリクエストを受け取る．そして，レスポンスもソケット通信を通して返す．

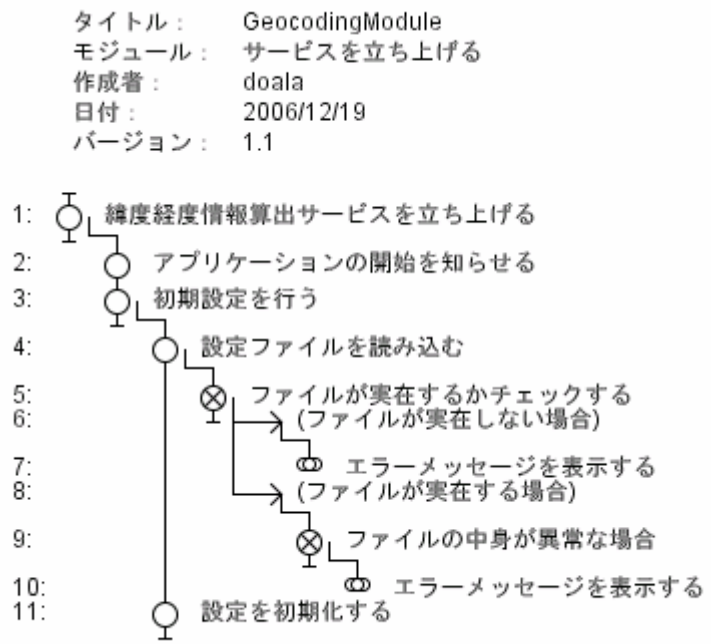
図 3

緯度経度情報は二つの数値（緯度と経度）のペアである．リクエスト内容によって，異なった地点の緯度経度情報を複数返すことがある．また，場合によってリクエストに該当する緯度経度情報がない場合もある．それぞれのケースのレスポンス形式を下記の表に記す．

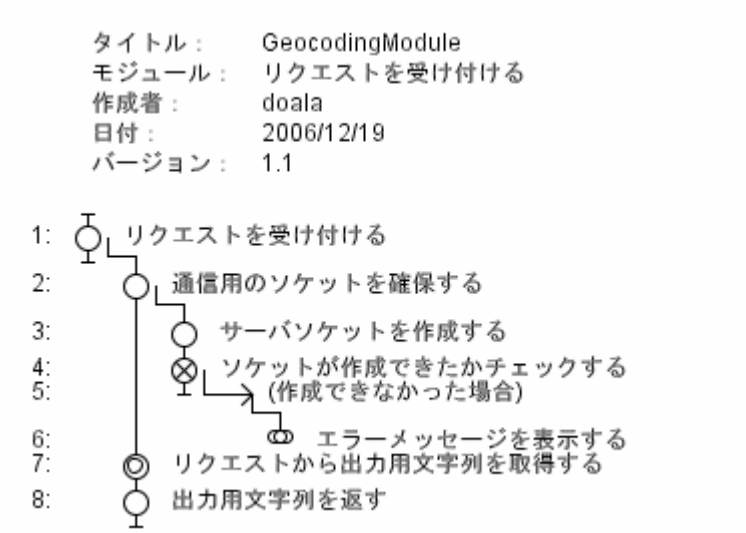
緯度経度情報の数	レスポンス形式
1 個	String 型（“{緯度},{経度}”） ※　緯度経度情報をコンマ区切りで返す
n 個	String 型（“{緯度 1},{経度 1};{緯度 2},{経度 2};{緯度 3},{経度 3};…{緯度 n},{経度 n}”） ※　セミコロン区切りで異なった地点の緯度経度情報を分ける
0 個	String 型　空文（””） ※　リクエストに該当する緯度経度情報が見つからなかった場合，レスポンスとして空文を返す

7. プログラム処理概要

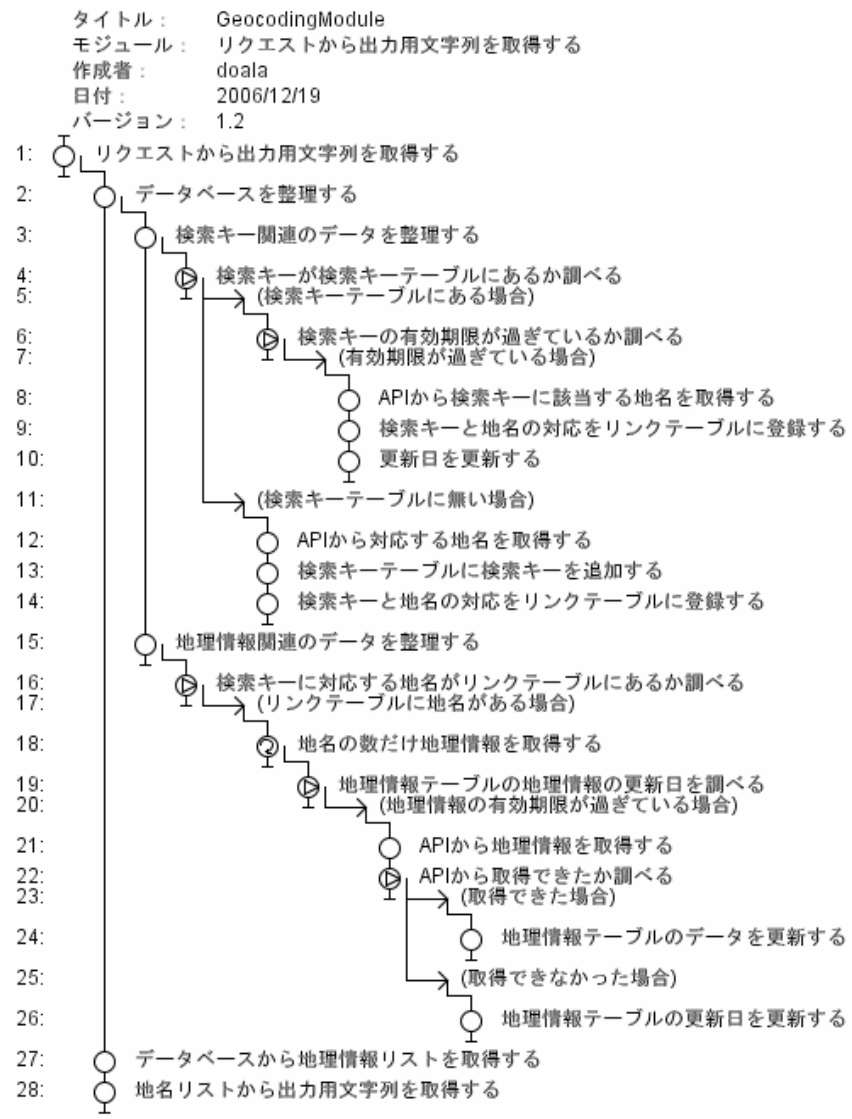
GeocodingModule を設計するに当たって，最初にプログラムの大まかな処理概要を HCP チャートに整理した．作成した HCP チャートは下記の三つである．



GeocodingModule が提供する緯度経度情報算出サービスに備えて設定の初期化を行っている．アプリケーションの開始を知らせた後，設定ファイルからの情報をプログラムに読み込んでいる．その設定ファイルが無い時や，中身が壊れていた場合はエラーメッセージをログに書き込みプログラムを正常に終了させる．



リクエストを受け取る準備を行い，通信用のソケットを確保．途中で例外が起きた場合，エラーをシステムログに書き込んだ後正常にプログラムを終了させる



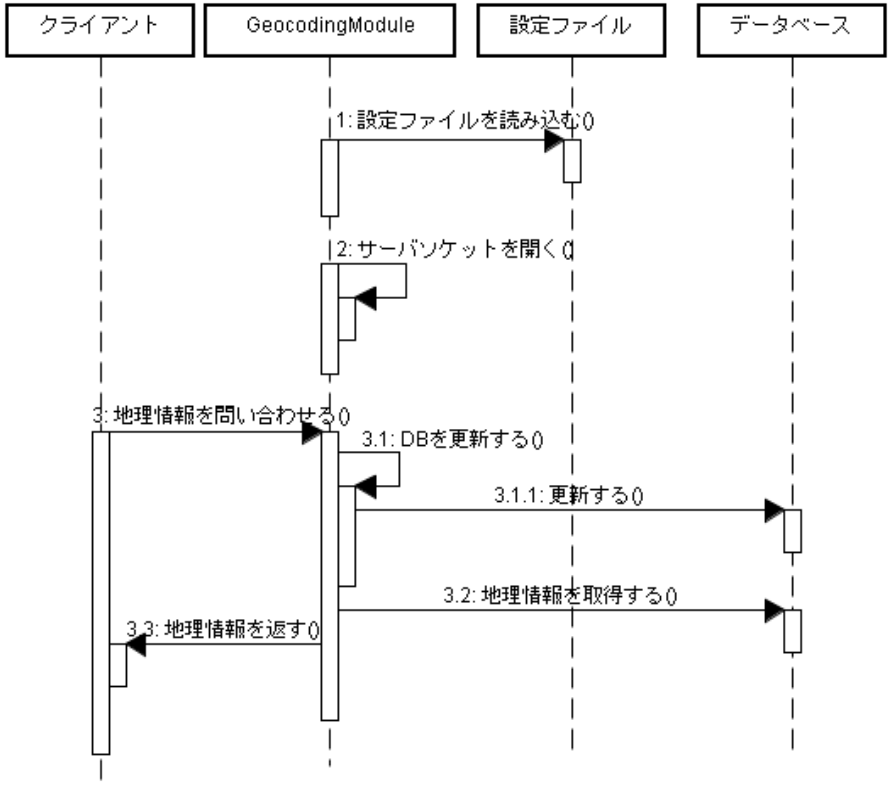
上の HCP チャートは GeocodingModule の骨格部分である，緯度経度算出サービスの動作概要を詳細に記載している．基本的な動作として，最初に渡された地名（検索キー）に該当する緯度経度情報が緯度経度情報キャッシュテーブル登録されているかどうか確認する．登録されている場

合は，そのままデータベースの情報をレスポンスとして返す．登録されて無い場合は，Web 上の緯度経度算出サービスを使用して緯度経度を調べる．そして，新しく取得した緯度経度情報を新たにデータベースに追加し，それをレスポンスとしてクライアントに返す．また，検索キーおよび地理情報の有効期限が切れていた場合は Web 上のサービスを使って随時最新の状態まで更新するようになっている．

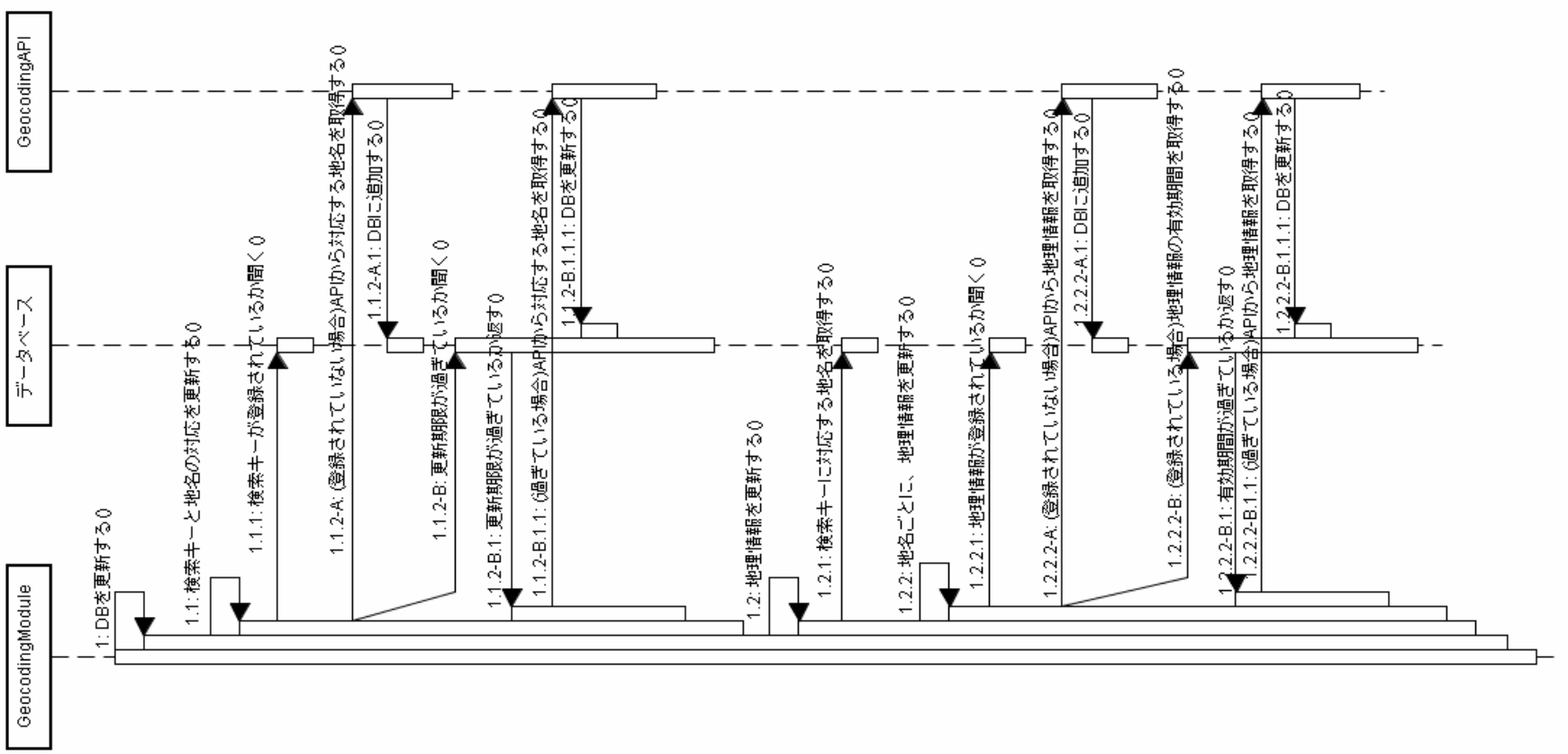
8. シーケンス図

GeocodingModule の動作を下記にある二つのシーケンス図で示す.

・シーケンス図（概略）

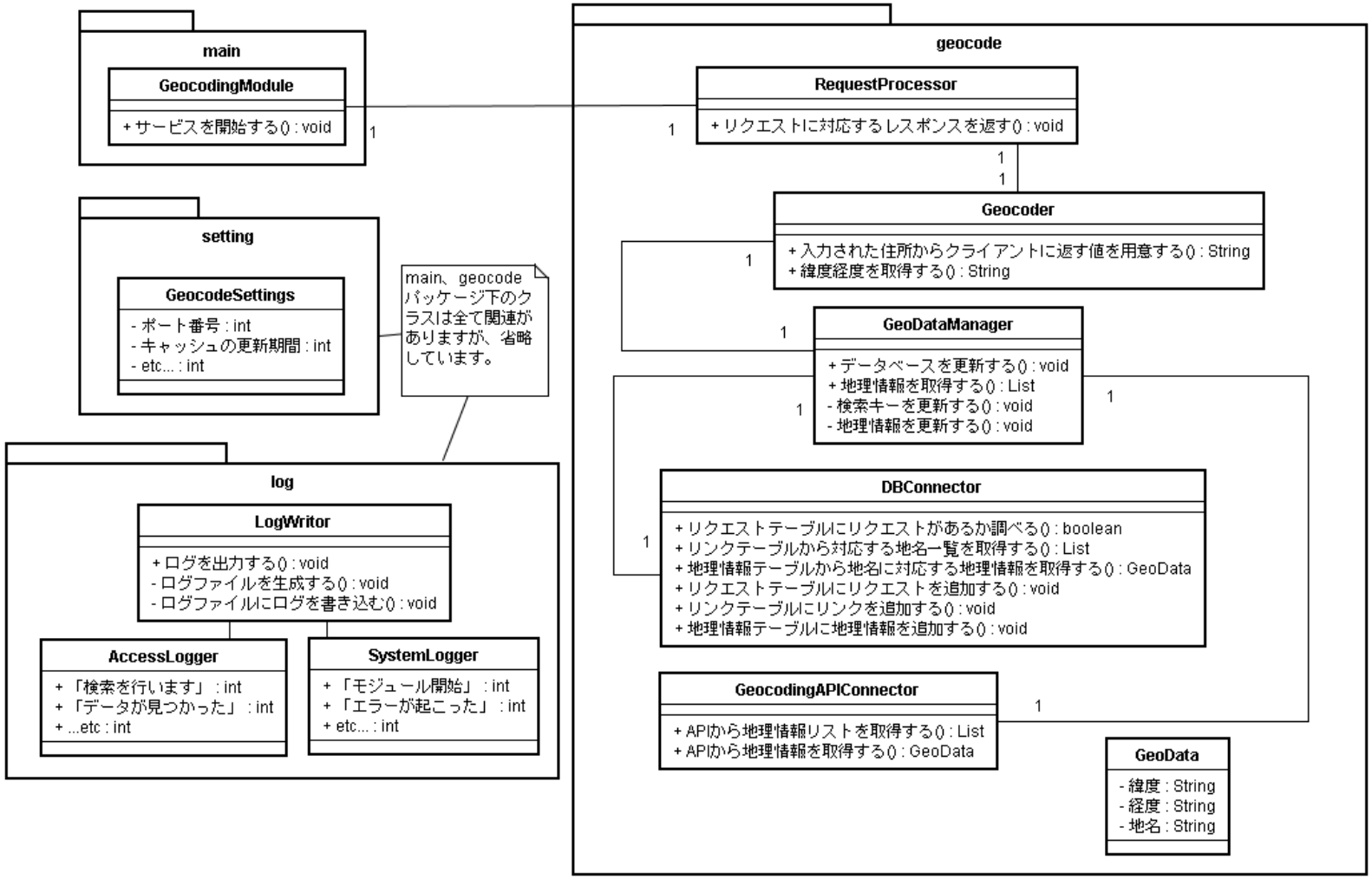


・ シーケンス図（DB を更新する部分）



9. クラス図

本モジュールにおけるクラス図は以下のようになっている.



上記のクラス図を見て分かるように，GeocodingModule は四つのパッケージから構成されている．それぞれのパッケージの役割は以下のようになっている．

パッケージ名	説明
main	緯度経度算出サービスの全面的管理
setting	プログラム設定の管理
log	システムログとアクセスログの管理
Geocode	地名を緯度経度に変換する GeocodingModule のコア部分 データベースと GeocodingAPI の接続を総括管理

続いて各パッケージ内のクラスについて詳しく解説する．

・ main パッケージ

クラス名	説明
GeocodingModule	緯度経度算出サービスの立ち上げまでの処理を管理するクラス

・ setting パッケージ

クラス名	説明
GeocodeSettings	設定ファイルの読み込みから始め，プログラム設定の管理するクラス 全てのパッケージと関連を持つ

・ log パッケージ

クラス名	説明
LogWriter	全面的なログ出力を行うクラス 全てのパッケージと関連を持つ
SystemLogger	システムログを管理するクラス

	LogWriter のインスタンスを持つ
AccessLogger	アクセスログを管理するクラス LogWriter のインスタンスを持つ

・ geocode パッケージ

クラス名	説明
APIManager	GeocodingAPI との通信を行うクラス
GeoDataManager	地名から緯度経度情報を取得するクラス
RequestProcessor	1 つのスレッドを表すクラス
DBManager	データベースとの通信を行うクラス
GeoData	緯度経度情報のオブジェクト
Geocoder	地理情報を取得するクラス

10. 緯度経度情報キャッシュテーブル

緯度経度情報キャッシュテーブルに GeocodingModule が扱う緯度経度情報をすべて格納する．Geocoding API で検索した緯度経度情報もここに登録する．キャッシュの期限を設けることによって，定期的にデータベースの中身を更新する．具体的なデータベース設計は下記に示す．

使用するデータベース：	MySQL 5.0.24
文字エンコード：	UTF-8

テーブル名： geocodingRequest

説明： GeocodingModule が受け取ったリクエスト内容を登録するデータベース．

テーブル設計：

No.	列英名	列名	型	桁数
1	searchQuery	検索キー	char	100
2	requestTimes	リクエスト回数	int	-
3	lastUpdatedRequest	最終更新日	Date	-

テーブル名： geocodingLink

説明： 検索キーと地名の対応付けを示すデータベース．

テーブル設計：

No.	列英名	列名	型	桁数
1	searchQuery	検索キー	char	100
2	locationName	リクエスト回数	Int	-
3	Rank	優先順位	Int	-

テーブル名：　geocodingData

説明：　地名とそれに該当する緯度経度情報が登録されているデータベース.

テーブル設計：

No.	列英名	列名	型	桁数
1	locationName	地名	Char	100
2	Latitude	緯度	Char	100
3	Longitude	経度	Char	100
4	insertDate	登録日時	Date	-
5	lastUpdated	更新日時	Date	-

データ挿入例：

テーブル：geocodingRequest

searchQuery	locationName	lastUpdatedRequest
東京	4	2007-01-15
三田	6	2007-01-15
江戸	2	1970-01-01

テーブル：geocodingLink

searchQuery	requestTimes	rank
東京	東京	1
三田	三田駅（東京）	1
三田	三田駅（東京）	2
江戸	江戸	1

テーブル：geocodingData

locationName	Latitude	Longitude	insertDate	lastUpdated
東京	100	200	2007-01-15	2007-01-15
三田駅（東京）	150	200	2007-01-15	2007-01-15
三田駅（兵庫）	200	300	2007-01-15	2007-01-15
江戸	100	200	1970-01-01	1970-01-01

※　同名異位置点の緯度経度情報は該当性順位を付けて，それぞれ別レコードとしてデータベースに登録される．

※　検索用語に該当する緯度経度情報が Geocoding API を用いても見つからなかった場合，null 値としてデータベースに登録される．

11. 設定ファイル

GeocodingModule の動作を制御する設定ファイルは別ファイルとして用意する．下記に具体的な仕様を示す．

設定ファイル名：	geoConfig.prop
ファイル形式：	Prop
文字エンコード：	UTF-8

設定ファイル仕様：

・全般

項目名	説明
modulePort=8080	GeocodingModule が使用するポート番号
requestEncode=UTF-8	GeocodingModule が TCP/IP ソケット通信を行う際に使用する文字エンコード
requestWaitTime=5000	API 接続のインターバル時間（ミリ秒単位）
returnMax=2	GeocodingModule が返す結果の数
cacheExpiration=15	緯度経度情報の更新期間（日単位）
searchKeyExpiration=15	検索キーの更新期間（日単位）
apiURL=http://www.geocoding.jp/api/?q=	Web 上の緯度経度算出サービスの URL
apiEncode=UTF-8	Geocoding API にリクエストを送信するときに使用する文字エンコード

・ログ関連の設定項目

項目名	説明
logEncode=UTF-8	ログのエンコード
logLevel=DEBUG	ログに記載するモジュールの動作内容の出力設定
systemLogPath=./systemLog/	システム関連のログの出力先

accessLogPath=./accessLog/	アクセスログの出力先

・データベース関連の設定項目

項目名	説明
DBURL=jdbc:mysql:///GeocodingModuleDB?useUnicode=true&characterEncoding=UTF8	データベースの URL
DBUserName=root	データベースのユーザ名
DBPass=pass	データベースユーザのパス

12. システムログ

GeocodingModule の動作をシステムログとして出力する.

ログのレベルにはエラー（致命的なエラーが発生したことを知らせる），警告（致命的ではないが意図通り動かない事態が発生したことを知らせる），情報（起動や設定ファイル読み込みといった情報を知らせる），デバッグの 4 レベルがあり，設定ファイルによって情報レベルを出力するかしないかを決定できる.

具体的なシステムログ仕様を下記に示す.

ファイル名：	systemLog_[yyyymmdd].txt
ファイル形式：	Txt
文字エンコード：	UTF-8

システムログの出力例

```
2007/01/26 00:53:49 INFO: starting service
2007/01/26 00:53:49 ERROR: server socket error
Address already in use: JVM_Bind
2007/01/26 03:29:55 DEBUG: Request recieved:ああ from/127.0.0.1
2007/01/26 03:29:55 DEBUG: Try to get geodata
2007/01/26 03:29:56 DEBUG: db connection has opened
2007/01/26 03:29:56 DEBUG: Try to update DB
2007/01/26 03:29:56 DEBUG: Try to update SearchKey Table
2007/01/26 03:29:56 DEBUG: Finished update SearchKey Table
2007/01/26 03:29:56 DEBUG: Try to update GeoData Table
2007/01/26 03:29:56 DEBUG: Finished update GeoData Table
2007/01/26 03:29:56 DEBUG: Finished to DB finished
```

```
2007/01/26 03:29:56 DEBUG: Try to get GeoDatas from DB
2007/01/26 03:30:03 DEBUG: db connection is closing.
2007/01/26 03:30:03 DEBUG: Get geodata succeed:null
2007/01/26 03:30:03 DEBUG: Returning result:null
2007/01/26 03:30:03 DEBUG: Succeed returning result:null
2007/01/26 03:30:17 DEBUG: Request recieved:アメリカ from/127.0.0.1
2007/01/26 03:30:17 DEBUG: Try to get geodata
2007/01/26 03:30:17 DEBUG: db connection has opened
2007/01/26 03:30:17 DEBUG: Try to update DB
2007/01/26 03:30:17 DEBUG: Try to update SearchKey Table
2007/01/26 03:30:17 DEBUG: Access API to get Location names
2007/01/26 03:30:18 DEBUG: got location:アメリカ
2007/01/26 03:30:18 DEBUG: Finished update SearchKey Table
2007/01/26 03:30:18 DEBUG: Try to update GeoData Table
2007/01/26 03:30:18 DEBUG: Access API to get GeoData
2007/01/26 03:30:19 DEBUG: Finished Access API succeed:true
2007/01/26 03:30:19 DEBUG: Finished update GeoData Table
2007/01/26 03:30:19 DEBUG: Finished to DB finished
2007/01/26 03:30:19 DEBUG: Try to get GeoDatas from DB
2007/01/26 03:30:19 DEBUG: db connection is closing.
2007/01/26 03:30:19 DEBUG: Get geodata succeed:39.077675,-98.425885
2007/01/26 03:30:19 DEBUG: Returning result:39.077675,-98.425885
2007/01/26 03:30:19 DEBUG: Succeed returning result:39.077675,-98.425885
```

13. アクセスログ

GeocodingModule が受けたリクエストの内容をログファイルとして出力することが出来る. 出力の可否は設定ファイルの中で設定することが出来る.

具体的なアクセスログ仕様を下記に示す.

ファイル名：	accessLog_[yyyymmdd].txt
ファイル形式：	Txt
文字エンコード：	UTF-8

アクセスログの出力例

```
2007/01/26 03:17:43 INFO: INFO received request  :ああ
2007/01/26 03:17:51 INFO: INFO no entry(s) for  :ああ
2007/01/26 03:29:55 INFO: INFO received request  :ああ
2007/01/26 03:30:03 INFO: INFO no entry(s) for  :ああ
2007/01/26 03:30:17 INFO: INFO received request  :アメリカ
2007/01/26 03:30:19 INFO: INFO found entry(s) for  :アメリカ
2007/01/26 03:31:09 INFO: INFO received request  :清水
2007/01/26 03:31:09 INFO: INFO found entry(s) for  :清水
2007/01/26 03:31:21 INFO: INFO received request  :清水
2007/01/26 03:31:21 INFO: INFO found entry(s) for  :清水
2007/01/26 03:31:37 INFO: INFO received request  :三田駅
2007/01/26 03:31:37 INFO: INFO found entry(s) for  :三田駅
2007/01/26 03:32:37 INFO: INFO received request  :清水駅
```

```
2007/01/26 03:32:46 INFO: INFO found entry(s) for  :清水駅
2007/01/26 03:32:52 INFO: INFO received request  :清水駅
2007/01/26 03:32:52 INFO: INFO found entry(s) for  :清水駅
2007/01/26 08:20:28 INFO: INFO received request  :東京タワー
2007/01/26 08:20:30 INFO: INFO found entry(s) for  :東京タワー
2007/01/26 08:24:19 INFO: INFO received request  :清水駅
2007/01/26 08:24:26 INFO: INFO found entry(s) for  :清水駅
2007/01/26 08:27:20 INFO: INFO received request  :東京
2007/01/26 08:27:21 INFO: INFO found entry(s) for  :東京
2007/01/26 08:27:30 INFO: INFO received request  :三田駅
2007/01/26 08:27:30 INFO: INFO found entry(s) for  :三田駅
2007/01/26 08:27:40 INFO: INFO received request  :府中
2007/01/26 08:27:41 INFO: INFO no entry(s) for  :府中
2007/01/26 08:27:53 INFO: INFO received request  :べなん
2007/01/26 08:27:55 INFO: INFO found entry(s) for  :べなん
2007/01/26 08:30:22 INFO: INFO received request  :魁
2007/01/26 08:30:24 INFO: INFO no entry(s) for  :魁
2007/01/26 08:30:33 INFO: INFO received request  :岡崎市
2007/01/26 08:30:33 INFO: INFO found entry(s) for  :岡崎市
```

14. エラー処理概要

GeocodingModule の動作中にエラーが発生した場合，以下のような対処をとる．その際，エラーメッセージをシステムログに出力する．

エラータイプ	エラーメッセージ	エラー内容	エラー処理
ERROR	Config file does not exist	設定ファイルが見つからない	プログラム終了
ERROR	Config file is broken	設定ファイルが壊れている	プログラム終了
ERROR	Cannot read config file	設定ファイルを読み込めない	プログラム終了
ERROR	Server socket error	サーバソケットエラー	プログラム終了
ERROR	Cannot connect to database	キャッシュテーブルの接続にできない	レスポンスとして null を返す
ERROR	Cannot disconnect from database	キャッシュテーブルの接続を切れない	
ERROR	Cannot read from input stream	入力ストリームから読み込めない	レスポンスとして null を返す
ERROR	Cannot write to output stream	出力ストリームに書き込めない	
ERROR	Cannot insert data	キャッシュテーブルにデータを挿入できない	
WARNING	Bad request	不正なリクエストを受け取った	レスポンスとして null を返す
WARNING	Cannot connect to API	Geocoding API に接続できない	レスポンスとして null を返す

V.　テスト仕様

1.　単体テスト

今回 g-mod が作成した緯度経度算出システムがクライアントの要求する機能が満たされているかを,緯度経度算出システム単体でのテストをする.

1) 単体テスト対応表

単体テストにおいて試験する仕様と，それに対応するテストは以下のようになっている．

システムの実行する処理の流れとについては設計書に記述したものを参照．

仕様分類	仕様名	正常処理テスト番号	異常処理テスト番号
起動	サービスを立ち上げる	1.1.1	2.1.1
設定	設定ファイルを読み込む	1.2.1，1.2.2	2.2.1，2.2.2
ソケット通信	サーバソケットを作成する	1.3.1	2.3.1
本処理	リクエストから出力用文字列を取得する	1.4.1，1.5.1	2.4.1，2.5.1
	DB を更新する	1.5.2.1	2.5.2，2.5.3
	DB から地理情報を取得する	1.5.2.2	2.5.2
DB 関連	検索キーが DB にあるか調べる	1.5.3.1.1	2.5.2
	検索キーの期限が切れているか調べる	1.5.3.1.2	2.5.2
	検索キーを更新する	1.5.3.1.3	2.5.2
	検索キーを新規に追加する	1.5.3.1.4	2.5.2
	検索キーの検索回数を増やす	1.5.3.1.5	2.5.2
	検索キーに対応する地名リストを取得する	1.5.3.2.1	2.5.2
	地名に対応する地理情報が DB にあるか調べる	1.5.3.2.2	2.5.2
	地理情報の有効期限を調べる	1.5.3.2.3	2.5.2
	地理情報を更新する	1.5.3.2.4	2.5.2
	地理情報を新規に追加する	1.5.3.2.5	2.5.2
	検索キーに対応する地理情報を全て取得する	1.5.3.3.1	2.5.2
API 関連	API から検索キーに対応する地名一覧を取得する	1.5.4.1	2.5.3
	API から地理情報を取得する	1.5.4.2	2.5.3

2) テスト仕様一覧

単体テストの仕様一覧は以下のようになっている.

1. 正常動作

システムが正常に動作している際に，仕様通りの動作をしているかをテストする.

項目は以下のようになっている.

分類番号	分類	テスト番号	テスト内容
1.1	起動	1.1.1	正常にサービスが立ち上がるか
1.2	設定ファイル	1.2.1	Property ファイルが正しい形式で存在する場合，読みこめているか
		1.2.2	property ファイルから正しいデータが取得できているか確認するか
1.3	ソケット通信	1.3.1	ソケット通信の確保ができるか
1.4	入力	1.4.1	引き渡されたデータを扱うことができるか
1.5	地理情報取得	1.5.1	リクエストを受け，それに対応する文字列が作成できるか
1.5.2	処理詳細	1.5.2.1	データベースを正しく更新するか
		1.5.2.2	データベースから地理情報が取得できるか
1.5.3.1	DB(検索キー)	1.5.3.1.1	検索キーが検索キーテーブルにあるか取得できるか
		1.5.3.1.2	検索回数を更新できるか
		1.5.3.1.3	検索キーの有効期限が切れているかどうか判定できるか
		1.5.3.1.4	検索キーを更新できるか
		1.5.3.1.5	検索キーを追加できるか
1.5.3.2	(地理情報)	1.5.3.2.1	検索キーに対応する地名一覧が取得できるか
		1.5.3.2.2	地名に対応する地理情報が地理情報テーブルに存在するか
		1.5.3.2.3	地理情報の有効期限が切れているかどうか判定できるか
		1.5.3.2.4	地理情報を更新できるか
		1.5.3.2.5	地理情報を追加できるか
1.5.3.3	(全般)	1.5.3.3.1	検索キーに対応する地理情報を取得できるか

1.5.4	API 関連	1.5.4.1	検索キーに対応する地名一覧が取得できるか
		1.5.4.2	地名に対応する地理情報を取得できるか
1.6	ログ関連	1.6.1	ログのパスが正しいか判定できるか
		1.6.2	ログが出力できるか

2.異常動作

システムが正常に動作していない際に，仕様通りの動作をしているかをテストする．

項目は以下のようになっている．

分類番号	分類	テスト番号	テスト内容
2.1	起動	2.1.1	起動できなかった場合，プログラムログにエラーを出力するか.
2.2	設定ファイル	2.2.1	設定ファイルが存在しない場合，プログラムログにエラーを出力し，プログラムを中止するか
		2.2.2	設定ファイルからデータが取得できなかった場合，プログラムログにエラーを出力し，プログラムを中止するか
2.3	ソケット通信	2.3.1	ソケット通信の確保できなかった場合，プログラムログにエラーを出力するか
2.4	入力	2.4.1	引き渡されたデータが扱えないものの場合，リクエストログに出力し，空データを返すか
2.5	処理	2.5.1	引き渡されたデータごとに地理情報を取得できなかった場合，リクエストログに出力し，空データを返すか
		2.5.2	DB にアクセスできない場合，プログラムログに出力し，空データを返すか
		2.5.3	API にアクセスできない場合，プログラムログに出力し，空データを返すか

3) テスト仕様詳細

1. 正常系

テスト番号	1.1.1		
テスト名称	正常にサービスが立ち上がるか		
入力形式	無し		
出力形式	サービスが立ち上がる		
テストケース番号	目的	入力	出力
1	サービスを立ち上げる	無し	サービスが立ち上がっている

テスト番号	1.2.1		
テスト名称	Property ファイルが正しい形式で存在する場合，読みこめているか		
入力形式	無し		
出力形式	boolean		
テストケース番号	目的	入力	出力
1	ファイルが存在する場合、読み込めることを返す	無し	TRUE
2	ファイルが存在しない場合、読み込めないことを返す	無し	FALSE
3	項目が欠けた状態で存在する場合、読み込めないことを返す	無し	FALSE

テスト番号	1.2.2		
テスト名称	property ファイルから正しいデータが取得できているか確認するか		
入力形式	無し		
出力形式	boolean		
テストケース番号	目的	入力	出力
1	読み込んだ値が正常な場合、読み込めることを返す	無し	TRUE
2	読み込んだ値が正常でない場合、読み込めないことを返す	無し	FALSE

テスト番号	1.3.1		
テスト名称	ソケット通信の確保ができるか		
入力形式	無し		
出力形式	ソケットが立ち上がる		
テストケース番号	目的	入力	出力
1	ソケットを立ち上げる	無し	ソケットが立ち上がっている

テスト番号	1.4.1		
テスト名称	引き渡されたデータを扱うことができるか		
入力形式	String		
出力形式	boolean		
テストケース番号	目的	入力	出力
1	扱うことが出来る場合、TRUE を返す	″東京″	TRUE
2	扱うことが出来ない場合、FALSE を返す	null	FALSE

テスト番号	1.5.1		
テスト名称	リクエストを受け、それに対応する文字列が作成できるか		
入力形式	String(検索キー)		
出力形式	List<地理情報データ>		
テストケース番号	目的	入力	出力(List の中身)
1	地理情報が 1 つ取得できる場合、指定した形式に沿って返す	"東京"	"100,200"
2	地理情報が複数取得できる場合、指定した形式に沿って返す	"三田"	"100,200;200,300"
3	取得できない場合、null を返す	"あああああ"	null

テスト番号	1.5.2.1		
テスト名称	データベースを正しく更新するか		
入力形式	String(検索キー)		
出力形式	DB が更新される		
テストケース番号	目的	入力	出力(DB の中身)
1	新規に検索したキーの場合、検索キーが追加される	"横浜"	
2	検索キーの更新期間が過ぎている場合、検索キーを更新する	"江戸"	
3	更新期間が過ぎていない場合、検索回数のみを更新する	"東京"	
4	地理情報が DB に無い場合、地理情報を取得し追加する	"横浜"	
5	地理情報の更新期間が過ぎている場合、地理情報を更新する	"江戸"	
6	地理情報の更新日が過ぎていない場合、更新をしない	"東京"	

テスト番号	1.5.2.2		
テスト名称	データベースから地理情報が取得できるか		
入力形式	String(検索キー		
出力形式	List<GeoData>		
テストケース番号	目的	入力	出力
1	地理情報が 1 つ取得できる場合、1 つ取得できる	″東京″	″東京″の地理情報
2	地理情報が複数取得できる場合、複数取得できる	″三田″	″三田（東京）″と″三田（兵庫）″の地理情報
3	地理情報が取得できない場合、空のリストが取得できる	″あああああ″	何も取得できない

テスト番号	1.5.3.1.1		
テスト名称	検索キーが検索キーテーブルにあるか取得できるか		
入力形式	String(検索キー),回数		
出力形式	boolean		
テストケース番号	目的	入力	出力（回数）
1	キーがデータベースに存在する場合、True を返す	″東京″	TRUE
2	キーがデータベースに存在しない場合、False を返す	″横浜″	FALSE

テスト番号	1.5.3.1.2		
テスト名称	検索回数を更新できるか		
入力形式	String(検索キー)		
出力形式	DB の検索回数の値が変わる		
テストケース番号	目的	入力	出力
1	検索キーの検索回数を増やし、更新する	″東京″	東京の requestTimes が2になる

テスト番号	1.5.3.1.3		
テスト名称	検索キーの有効期限が切れているかどうか判定できるか		
入力形式	String(検索キー)		
出力形式	boolean		
テストケース番号	目的	入力	出力(リンク個数)
1	検索キーの期限が切れている場合、True を返す	"東京"	TRUE
2	検索キーの期限が切れていない場合、False を返す	"江戸"	FALSE

テスト番号	1.5.3.1.4		
テスト名称	検索キーを更新できるか		
入力形式	String(検索キー)、List<String>(地名)		
出力形式	DB が更新されている		
テストケース番号	目的	入力	出力(String□の中身)
1	検索キーの更新日と、リンクテーブルを更新する	"東京"、List("東京")	更新日とリンクテーブルが更新

テスト番号	1.5.3.1.5		
テスト名称	検索キーを追加できるか		
入力形式	String(検索キー)、List<String>(地名)		
出力形式	DB か更新されている		
テストケース番号	目的	入力	出力(List の中身)
1	地名がある場合、検索キーテーブルとリンクテーブルに追加する	"東京"、List("東京")	検索キー・リンクテーブルに追加
2	地名が無い場合、検索キーテーブルのみを更新する	"あああああ"、NULL	検索キーテーブルに追加

テスト番号	1.5.3.2.1		
テスト名称	検索キーに対応する地名一覧が取得できるか		
入力形式	String(検索キー)		
出力形式	List<地名>		
テストケース番号	目的	入力	出力
1	検索キーに対応する地名が 1 つの場合、1 つだけ返す	″東京″	List(″東京″)
2	検索キーに対応する地名が複数ある場合、全て返す	″三田″	List(″三田(東)″、″三田(兵庫)″)
3	検索キーに対応する地名が無い場合、NULL を返す	″あああああ″	List(null)

テスト番号	1.5.3.2.2		
テスト名称	地名に対応する地理情報が地理情報テーブルに存在するか返す		
入力形式	String(地名)		
出力形式	boolean		
テストケース番号	目的	入力	出力(DB より取得)
1	地理情報テーブルに存在する場合、True を返す	″三田″	TRUE
2	地理情報テーブルに存在しない場合、False を返す	″あああああ″	FALSE

テスト番号	1.5.3.2.3		
テスト名称	地理情報の有効期限が切れているかどうか判定できるか		
入力形式	String(地名)		
出力形式	boolean		
テストケース番号	目的	入力	出力
1	期限が切れている場合、True を返す	″江戸″	TRUE
2	期限が切れていない場合、False を返す	″東京″	FALSE

テスト番号	1.5.3.2.4		
テスト名称	地理情報を更新できるか		
入力形式	地理情報データ		
出力形式	地理情報テーブルが更新される		
テストケース番号	目的	入力	出力(DB の最終更新日)
1	地理情報が更新される	"江戸"、100、50	入力通りの地理情報が DB に記録される

テスト番号	1.5.3.2.5		
テスト名称	地理情報を追加できるか		
入力形式	String(地名)		
出力形式	地理情報テーブルが更新される		
テストケース番号	目的	入力	出力
1	地理情報を追加する	"カナダ"、50、20	入力通りの地理情報が DB に記録される

テスト番号	1.5.3.3.1		
テスト名称	検索キーに対応する地理情報を取得できるか		
入力形式	String(地名)		
出力形式	List<地理情報データ>		
テストケース番号	目的	入力	出力(List の中身)
1	地理情報が 1 つ存在する場合、1 つ入ったリストを返す	"東京"	東京
2	地理情報が複数存在する場合、複数入ったリストを返す	"三田"	三田(東京)、三田(兵庫)
3	該当するものが無い場合、1 つも入っていないリストを返す	"あああああ"	null

テスト番号	1.5.4.1		
テスト名称	検索キーに対応する地名一覧が取得できるか		
入力形式	String(地名)		
出力形式	List<String>		
テストケース番号	目的	入力	出力(リストの中身)
1	キーに対応する地名が 1 つある場合、地理情報を保存し、1 つ返す	"東京"	"東京"
2	キーに対応する地名が複数ある場合、全て返す	"三田"	"三田(東京)"、"三田(兵庫)"
3	対応する地名が無い場合、null を返す	"あああああ"	null

テスト番号	1.5.4.2		
テスト名称	地名に対応する地理情報を取得できるか		
入力形式	String(地名)		
出力形式	List<地理情報データ>		
テストケース番号	目的	入力(List の中)	出力
1	保存してある場合、保存されている所から取得する	東京	"100,200"
2	保存されていない場合、保存	三田(東京)、三田(兵庫)	"100,200;200,300"
3	存在しない場合、null を出力する	null	null

テスト番号	1.6.1		
テスト名称	ログのパスが正しいか判定できるか		
入力形式	String(ログのパス)		
出力形式	boolean		
テストケース番号	目的	入力	出力
1	ログが出力可能なパスの場合、True を返す	./Log/	TRUE
2	ログが出力不能なパスの場合、False を返す	¥¥"¥¥¥"	FALSE

テスト番号	1.6.2		
テスト名称	ログが出力できるか		
入力形式	無し		
出力形式	ログファイル		
テストケース番号	目的	入力	出力
1	ログファイルが存在しない場合、新規に作成する	″起動″	ログが出力される
2	ログファイルが存在する場合、ファイル内に書き込む	″起動″	既存のファイルが上書きされる

2. 異常系

テスト番号	2.1.1		
テスト名称	起動できなかった場合，プログラムログにエラーを出力するか．（1.1.1 参照）		
入力形式	無し		
出力形式	プログラムログに出力する		
テストケース番号	目的	入力	出力（プログラムログ）
1	立ち上がらない場合、プログラムログに出力する	無し	「立ち上がらなかった」

テスト番号	2.2.1		
テスト名称	設定ファイルが存在しない場合，プログラムログにエラーを出力し，プログラムを中止するか(1.2.1 参照）		
入力形式	無し		
出力形式	プログラムログに出力する		
テストケース番号	目的	入力	出力（プログラムログ）
1	設定ファイルが存在しない場合、プログラムログに書き込み、プログラムを中止する	無し	「設定ファイルが存在しない」

テスト番号	2.2.2		
テスト名称	設定ファイルからデータが取得できなかった場合，プログラムログにエラーを出力し，プログラムを中止するか(1.2.2 参照）		
入力形式	無し		
出力形式	プログラムログに出力する		
テストケース番号	目的	入力	出力（プログラムログ）
1	設定ファイルが形式どおりでない場合、プログラムログに書き込み、プログラムを中止する	無し	「設定ファイルが形式通りでない」

テスト番号	2.3.1		
テスト名称	ソケット通信の確保できなかった場合，プログラムログにエラーを出力するか（1.3.1 参照）		
入力形式	無し		
出力形式	プログラムログに出力する		
テストケース番号	目的	入力	出力（プログラムログ）
1	ソケットが確保できない場合、プログラムログに書き込む	無し	「ソケット確保失敗」

テスト番号	2.4.1		
テスト名称	引き渡されたデータが扱えないものの場合，リクエストログに出力し，空データを返すか（1.4.1 参照）		
入力形式	String		
出力形式	リクエストログ、String（null）		
テストケース番号	目的	入力	出力（リクエストログ、返り値）
1	扱えない場合、リクエストログに出力し、空データを返す	null	「null は扱えません」、null

テスト番号	2.5.1		
テスト名称	引き渡されたデータごとに地理情報を取得できなかった場合，リクエストログに出力、空データを返すか（1.5.2 参照）		
入力形式	無し		
出力形式	リクエストログ、String（null）		
テストケース番号	目的	入力	出力（リクエストログ、返り値）
1	取得できなかった場合、リクエストログに出力し、空データを返す	アフリカ	「アフリカに対応する地名が無いです」、null

テスト番号	2.5.2		
テスト名称	DB にアクセスできない場合，プログラムログに出力し，空データを返すか		
入力形式	String 等		
出力形式	プログラムログに出力する、String（null）		
テストケース番号	目的	入力	出力（プログラムログ、返り値）
1	DB が存在しない状態でアクセスした場合、ログに出力し、空データを返す	″東京″のリクエスト数取得	「DB に接続できません」、null

テスト番号	2.5.3		
テスト名称	API にアクセスできない場合，プログラムログに出力し，空データを返すか（1.5.15,1.5.16 参照）		
入力形式	String 等		
出力形式	プログラムログに出力する、String（null）		
テストケース番号	目的	入力	出力（プログラムログ、返り値）
1	API にアクセスできない状態でアクセスした場合、ログに出力し、空データを返す	″カナダ″を取得	「API に接続できません」、null

データベースの内容

geocodingRequest テーブル

searchQuery	requestTimes	lastUpdatedRequest
東京	1	2007/1/15
三田	1	2007/1/15
江戸	1	1970/1/1

geocodingLink テーブル

searchQuery	locationName	rank
東京	東京	1
三田	三田駅(東京)	1
三田	三田駅(兵庫)	2
江戸	江戸	1

geocodingData テーブル

locationName	latitude	longitude	insertDate	lastUpdated
東京	100	200	2007/1/15	2007/1/15
三田駅(東京)	100	200	2007/1/15	2007/1/15
三田駅(兵	200	300	2007/1/15	2007/1/15

庫）				
江戸	100	200	1970/1/1	1970/1/1
アフリカ	null	null	2007/1/15	2007/1/15

設定ファイルの内容

modulePort=8080

moduleTimeout=5000

databaseTimeout=5000

cacheExpiration=15

returnMax=2

requestURL=http://www.geocoding.jp/api/?q=

programLog=off

programLogPath=./program/

programLogEncode=UTF-8

requestLog=on

requestLogPath=./

requestLogEncode=UTF-8

requestEncode=UTF-8

requestWaitTime=5000

DBURL=jdbc:mysql:///GeocodingModuleDB?useUnicode=true&characterEncoding=UTF8

requestExpiration=15

DBUserName=root

DBPass=null

2. 結合テスト

今回 g-mod が作成した緯度経度算出システムが，DMC システムと接続した際に，両システムが共に正常な稼働する事ができるかをテストする．
テストは運用環境下にあるシステムを対象に行う．

1) テストシナリオ

結合テストの内容とテストシナリオは以下のようになっている.

分類番号	分類	テスト番号	テスト名	シナリオ番号	シナリオ内容	確認項目	
1	正常フロー	1.1	初回起動	1.1.1	DB に何も入っていない状態で, リクエストする	DB	DB に入力があること
						レスポンス	指定どおりの形式でレスポンスが行われたこと
		1.2	次回起動	1.1.1	新規情報をリクエストする	DB	DB に新規データ分の入力があること
						DB	DB に新規データ分以外は更新されていないこと
						レスポンス	指定どおりの形式でレスポンスが行われたこと
				1.1.2	既存情報をリクエストする	DB	DB のリクエスト回数以外は更新されていないこと
						レスポンス	指定どおりの形式でレスポンスが行われたこと
				1.1.3	複数該当情報をリクエストする	DB	リクエスト 1 つに対し、地名が複数登録されたこと
						レスポンス	指定どおりの形式でレスポンスが行われたこと
				1.1.4	該当しないリクエストをする	DB	検索キーのみが更新されていること
						レスポンス	null が帰ったこと
		1.3	地理情報更新	1.3.1	地理情報テーブルに更新期間の切れたデータがあり, それに対応するリクエストをする	DB	データが更新されていることを確認する
						レスポンス	指定どおりの形式でレスポンスが行われたこと
		1.4	リクエスト更新	1.4.1	検索キーに更新期間の切れたデータがあり, それに対応するリクエストをする	DB	データが更新されていることを確認する
						レスポンス	指定どおりの形式でレスポンスが行われたこと
2	異常フロー	2.1	設定異常 1	2.1.1	設定ファイルが存在しなかった場合	プログラム	終了する
						ログ	報告をする
		2.2	設定異常 2	2.2.1	設定ファイルが異常な形式の場合	プログラム	終了する
						ログ	報告をする

		2.3	入力異常	2.3.1	入力データの形式が異常の場合	レスポンス	null が返る
						ログ	報告をする
		2.4	DB 異常	2.4.1	DB の形式が異常の場合	レスポンス	null が返る
						プログラム	終了する
		2.5	API 異常	2.4.1	DB 接続が異常の場合	レスポンス	null が返る
		2.5	API 異常	2.5.1	API へ接続ができない場合	プログラム	終了する
						プログラム	終了する

VI. テスト結果

1. テスト結果について

・単体テストについて
テストは複数回に分けて行った。
各自が担当箇所のテストプログラムを作成し、それを基にコーディングを行った。
1 月 8 日に各自の担当箇所のマージ作業を行い、その際に全体で行った単体テストを第一回目の記録とした。
それ以前に各自の担当箇所で出たバグは計測していない。
その後、マージ作業によって生まれた不具合の修正や、及びその後の設計変更に伴うテストの変化があり、
最終的に 1 月 24 日に 2 回目の単体テストを行った。
既にマージしてある状態であり、各自のパーツの繋がりができているため、
ここで発見されたエラーは 1 つも無かった。

・結合テストについて
結合テストは、本番環境下において期待通りに動くかどうかを調べた。
テストを行ったのは 1 月 24 日である。
ローカル環境下で頻繁に実行及び単体テストをしているため、
予想通りの結果が返ることが確認できた。

2. 単体テスト結果

テスト結果一覧

テスト番号	テストケース番号	1 回目(1 月 8 日)	2 回目(1 月 24 日)
1.1.1	1	○	○
1.2.1	1	×	○
	2	×	○
	3	×	○
1.2.2	1	○	○
	2	○	○
1.3.1	1	○	○
1.4.1	1	○	○
	2	○	○
1.5.1	1	○	○
	2	○	○
	3	×	○
1.5.2.1	1	○	○
	2	×	○
	3	×	○
	4	○	○
	5	×	○
	6	×	○
1.5.2.2	1	○	○
	2	○	○
	3	○	○

1.5.3.1.1	1	○	○
	2	○	○
1.5.3.1.2	1	○	○
1.5.3.1.3	1	○	○
	2	○	○
1.5.3.1.4	1	×	○
1.5.3.1.5	1	○	○
	2	○	○
1.5.3.2.1	1	○	○
	2	○	○
	3	○	○
1.5.3.2.2	1	○	○
	2	○	○
1.5.3.2.3	1	○	○
	2	○	○
1.5.3.2.4	1	×	○
1.5.3.2.5	1	○	○
1.5.3.3.1	1	○	○
	2	○	○
	3	○	○
1.5.4.1	1	×	○
	2	○	○
	3	○	○
1.5.4.2	1	○	○
	2	×	○

	3	○	○
1.6.1	1	×	○
	2	×	○
1.6.2	1	×	○
	2	×	○
2.1.1	1	○	○
2.2.1	1	○	○
2.2.2	1	○	○
2.3.1	1	○	○
2.4.1	1	○	○
2.5.1	1	○	○
2.5.2	1	○	○
2.5.3	1	○	○
		42/58	58/58

3. 結合テスト結果

分類番号	分類	テスト番号	テスト名	シナリオ番号	シナリオ内容	確認項目		使用データ	テスト結果
1	正常フロー	1.1	初回起動	1.1.1	DB に何も入っていない状態で，リクエストする	DB	DB に入力があること	東京	○
						レスポンス	指定どおりの形式でレスポンスが行われたこと		○
		1.2	次回起動	1.1.1	新規情報をリクエストする	DB	DB に新規データ分の入力があること	京都	○
						DB	DB に新規データ分以外は更新されていないこと		○
						レスポンス	指定どおりの形式でレスポンスが行われたこと		○
				1.1.2	既存情報をリクエストする	DB	DB のリクエスト回数以外は更新されていないこと	東京	○
						レスポンス	指定どおりの形式でレスポンスが行われたこと		○
				1.1.3	複数該当情報をリクエストする	DB	リクエスト 1 つに対し、地名が複数登録されたこと	三田	○
						レスポンス	指定どおりの形式でレスポンスが行われたこと		○
				1.1.4	該当しないリクエストをする	DB	検索キーのみが更新されていること	てすと	○
						レスポンス	null が帰ったこと		○
		1.3	地理情報更新	1.3.1	地理情報テーブルに更新期間の切れたデータがあり， それに対応するリクエストをする	DB	データが更新されていることを確認する		○
						レスポンス	指定どおりの形式でレスポンスが行われたこと		○

		1.4	リクエスト更新	1.4.1	検索キーに更新期間の切れたデータがあり, それに対応するリクエストをする	DB	データが更新されていることを確認する		○
						レスポンス	指定どおりの形式でレスポンスが行われたこと		○
2	異常フロー	2.1	設定異常1	2.1.1	設定ファイルが存在しなかった場合	プログラム	終了する		○
						ログ	報告をする		○
		2.2	設定異常2	2.2.1	設定ファイルが異常な形式の場合	プログラム	終了する		○
						ログ	報告をする		○
		2.3	入力異常	2.3.1	入力データの形式が異常の場合	レスポンス	null が返る		○
						ログ	報告をする		○
		2.4	DB 異常	2.4.1	DB の形式が異常の場合	レスポンス	null が返る		○
						プログラム	終了する		○
		2.5	API 異常	2.4.1	DB 接続が異常の場合	レスポンス	null が返る		○
		2.5	API 異常	2.5.1	API へ接続ができない場合	プログラム	終了する		○
						プログラム	終了する		○

VII. インストール手順

1. 推奨環境

CPU	300MHz 以上
メモリ	128 RAM 以上
ハードディスク	1GB 以上の空き容量
その他	JRE 1.5 以上がインストール済み MySQL 5.0 以上がインストール済み

2. インストール手順

1) ファイル一覧

Geocoding Module の動作に必要なファイルを下記の表で記す。

ファイル名	説明
GeocodingModule.jar	地名から緯度経度情報を割り出すモジュールの本体
geoConfig.prop	Geocoding Module 用の設定ファイル GeocodingModule.jar と同じディレクトリにおく必要がある
Lib フォルダ	MySQLドライバであるmysql-connector-java-5.0.4-bin.jarが格納されているフォルダ
GeocodingModule.sh	GeocodingModule を呼び出すスクリプトファイル
Data2.sql	DB の初期データ SQL ファイル

3. モジュールインストール方法

Zip ファイルを解凍すると以下のような配置でファイルが出力されます。

```
GeocodingModule.jar geoConfig.prop GeocodingModule.sh lib data2.sql
```

上記のファイルを任意のディレクトリを配置する。

4. データベース作成

地理情報を保存するためのデーベースを作成する必要があります。作成方法は以下のようになります。

```
% mvsql –u root –p      //root でログイン
% Enter password:       //password を入力する（初回インストール時には無し）

mysql> CREATE DATABASE geocodingmoduledb;          //DB 作成する。DB 名は任意
```

5. 初期データ挿入

初期データのあるディレクトリで以下のコマンドを使いデータベースに挿入します。

```
% mvsql [DB名] < [SQLファイル] -u [DBユーザ名] –p [DBパスワード]
```

例

```
% mvsql geocodingmoduledb < data2.sql -u root –p
```

6. PATH設定

GeocodingModule を起動するための PATH の設定を行います。

/etc/profile に以下のラインを追加します

```
export G_HOME=[GeocodingModuke.shの置いてある任意のPATH名]
export PATH="$PATH:$G_HOME":
例
export G_HOME=/home/g-mod/GeocodingModule
export PATH="$PATH:$G_HOME":
```

GeocodingModule.sh の権限を変更する必要があります。

```
% chmod 777 GeocodingModule.sh
```

以上でインストールは完了です。

VIII. 使用手順

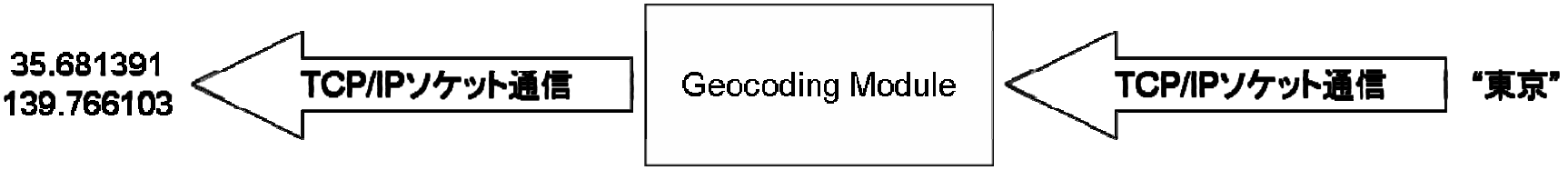
1. 起動手順

Gecoding Module はコンソールで次のコマンドを実行することで起動することが出来る。

```
% GeocodingModule.sh
Geocoding Module started
```

2. 利用手順

Geocoding Module を起動した状態で、TCP/IP のソケット通信を通してモジュールに地名を送信するとそれに該当する緯度経度情報をソケット通信で返してくる。



3. 終了手順

Geocoding Module を終了するためには kill コマンドで強制的に終了させる。

```
% kill [Geocoding Module のプロセス ID]
```

4. 設定ファイル項目

下記に設定ファイルの項目内容について記す。

1) 全般の設定項目

`modulePort=8080`

Geocoding Module が使用する任意のポート番号（1 から 65535 までの整数)。既に使用中のポート番号は利用できない。

`requestEncode=UTF-8`

Geocoding Module が TCP/IP ソケット通信を行う際に使用する文字エンコード。通常は UTF-8。

`requestWaitTime=5000`

連続して Geocoding API に地理情報を問い合わせる場合、次のリクエストを出すまでの待ち時間。
ミリ秒単位で指定する。通常は 5000 ミリ秒。

`returnMax=2`

Geocoding Module が返す緯度経度情報の最大件数。

`cacheExpiration=15`

地理情報の有効期限。日単位で指定する。

`searchKeyExpiration=15`

検索キーの有効期限。日単位で指定する。

```
apiURL=http://www.geocoding.jp/api/?q=
```

使用する外部の緯度経度算出サービスの URL。

Google 社の Geocoding API を使用するため、この項目内容は変更しない。

```
apiEncode=UTF-8
```

Geocoding API にリクエストを送信するときに使用する文字エンコード。

2) ログ関連の設定項目

```
logEncode=UTF-8
```

システムログとアクセスログの文字エンコード。

```
logLevel=DEBUG
```

ログに記載するモジュールの動作内容の出力設定。ERROR・WARNING・INFO・DEBUG の 4 種類まで設定可能。

- ERROR：重大なエラー情報だけを出力する。
- WARNING：ERROR に加え、警告メッセージを出力する。
- INFO：上記の二つに加え、重要な情報も出力する。
- DEBUG：上記の三つに加え、詳細なデバッグ情報を全て出力する。

```
systemLogPath=./systemLog/
```

システムログの出力パス。

```
accessLogPath=./accessLog/
```

アクセスログの出力パス。

3) データベース関連の設定項目

```
DBURL=jdbc:mysql:///GeocodingModuleDB?useUnicode=true&characterEncoding=UTF8
```

データベースの URL。基本的にこの項目内容は変更しない。

```
DBUserName=root
```

データベースに接続するためのユーザー名。通常は root。

```
DBPass=
```

データベースに接続するためのパスワード。通常は空文。

IX. プロジェクト経緯

1. 要件定義

1. 実施期間

2006 年 10 月 5 日～2007 年 2 月 23 日

2. 作業経緯

要件定義の目的は，g-mod チームが作るべきものをはっきりさせることである．

プロジェクト結成前の 10 月 4 日に，PM の谷田部が嶋津先生と助手の斉藤さん，戸並さんを交えた要件ヒアリングを行っており，その際に要求仕様書と機能仕様書を頂いていた．

プロジェクト結成時には PM からメンバーに対しての説明があり，その後の方針としては，説明を聞き，書類を読んでもわからなかった点や曖昧に感じた点をクライアントに質問するというものであった．

嶋津先生と遠峰様の 2 名を迎えて行われた 10 月 12 日のヒアリングでは，キャッシュのクリア期間や設定ファイルの有無，入出力のエンコードといった細かい仕様についての質問や，要求仕様書に記載されていた「知の再編」という言葉の定義などの疑問点を解消していった．このヒアリングの際に，嶋津先生から仕様が一部変更され，新たに仕様書を作成し次第渡すという言葉を頂いた．この通り，今後も仕様の変更が有り得るため，その際には必ず告知し仕様書を渡すとの約束を頂いた．

このヒアリングを受け，佐藤が要件定義書を兼ねたプロジェクト提案書の作成を行った．チーム内レビューを経て 10 月 28 日作成を完了した．また，新たな仕様書を読んで感じた疑問点等を一つの書類にまとめ，10 月 25 日に要件質問として遠峰様に送信した．

プロジェクト提案書の承認を貰うことが当面の目標となっていたが，嶋津先生と直接会う日をなかなか取ることができず，10 月 30 日にとりあえずメールで送信し，レビューして頂くということにした．返信を 11 月 7 日に頂き，その際修正が必要と言われた点を変更し，11 月 9 日に承認を頂こうということになった．

しかし，11 月 9 日に承認を貰うことはできなかった．その際，「要求仕様確認書」としてなら大丈夫であるが「プロジェクト提案書」としては承認することができないという言葉を頂いた．その理由として，プロジェクトの目的や目標が曖昧だということを指摘された．このプロジェクトは何をするプロジェクトであり，何を持って成功とするのかがわからないものであった．DMC の

一部として，言われた部分の作成をするということであればこれで良いが，g-mod チームとしての目的がわからない，と言われた．g-mod チームが，DMC の一部ではなく，大岩研究室にある 1 つのプロジェクトとして何ができるかと言うことを明確にし，提案をして欲しいとの言葉を頂いた．

その後，チーム内でプロジェクト提案についての討論を行い，11 月 20 日に遠峰様との詳細仕様の質問に続き，11 月 22 日に嶋津先生の元へヒアリングのため赴いた．その際，新しく何かを提案してくれるのは大歓迎であるが，それができるだけの証拠が欲しいということを言われた．また，現状のシステムの問題点を解消する GeocodingModule の役割は，これまで手動で緯度経度情報を登録していたのが自動化され速度・正確さ共に向上するということであり，何が問題点でなぜその提案によって解決できるのかを考えるようにという言葉を頂いた．

その後，11 月 30 日に嶋津先生とヒアリングを行った．しかし，この際に質問しようとしていた事柄は全て嶋津先生の管轄外であり，遠峰様に聞くべき内容であるとの言葉を頂いた．これは DMC の役割分担を理解していなかったためである．メンバーは「承認を頂くのは嶋津先生」「詳細の仕様を聞くのは遠峰様」というように考えていたが，実際は「DMC システム全体に関しては嶋津先生」「GeocodingModule に関しては遠峰様」という管轄になっていた．よって，この日は要件定義に関する有力な情報を得ることは出来なかった．

チーム内で目的と目標の定義を行った．最終的に，目的は「DMC で使ってもらえる新しいシステムを開発すること」ということ，目標は「高品質のシステムを構築すること」ということになり，成功したかどうかを判定する報酬として「DMC トップページに g-mod や CreW のロゴを表示する」，「コンテンツアップローダー権限を CreW が貰う」等，7 つ考えた．これを元に 12 月 6 日に遠峰様の元に持っていき，提案を行ったが，成功報酬に同意しにくい箇所があるため，承認はできないと言われた．

次の案を考えるため，12 月 7 日にミーティングを行う予定であったが，その際に嶋津先生と松澤さんから「交渉になっていない」との言葉を頂いた．報酬が何なら大丈夫なのか，何を目的として報酬を欲しがっているのか，そういったことが遠峰様には伝わっていなかったのではないかという指摘を頂いた．また，その際にあっさりと引きあげてきたのも問題であり，何を目的として話し合いを行おうとしたのかが明確になっていないという指摘を受けた．

この指摘を元に，その日決める事柄を記載した「ミーティング用レジュメ」を作成し，それを携えて 12 月 13 日に遠峰様との交渉を行った．この際，成功報酬が必要であると言うことには理解をして頂き，その内容はこの後考えるとして，それ以外の部分は大丈夫であるということで，ようやく合意を得ることができた．

その後は，合意を取ることができたということもあり，実装等の優先すべき作業があったため承認を得るのは後回しになってしまったが，年明けの 1 月 15 日にプロジェクト提案書に遠峰様に提示し，何も問題が無いということで，承認を得ることができた．

3. 反省・感想

このプロジェクトにおいて一番のネックになったのが要件定義であるのは間違いない．当初の予定では 10 月 5 日～10 月 19 日の予定であった．それが年明けまで長引いてしまった要因は幾つか考えられる．

まず，目標と目的を考えていなかったということ．当初は言われたものを作ればいいとしか考えておらず，目標は無く，目的は，強いて言えば「完成させる」というくらいのものであった．これでは DMC の一部であると言われても当然のことである．当初から具体的な要求が定義されていたと言うこともあり，自分たちはそれを満たせばいいとしか考えておらず，どのように要求定義書に記載されている事項をクリアしていこうかということしかプロジェクト提案書に記さなかった．現に，進捗報告や中間発表会では，DMC システムの説明ばかりを行い，g-mod チームの説明は全くと言っていいほど行わなかった．原因として，最初にかなりの具体性を持った要求定義書及び機能仕様書を渡され，何が何でもその通りに作成しなければならないという意識が全メンバーのあり，その考えにしばられてしまっていたということが考えられる．最終的には，明確な目標・目的の定義及び，成功報酬の交渉によって解決したが，この問題に気付くのに 1 月以上の時間がかかり，解決するのにも同等の時間が必要となった．十分に反省すべき点であろう．

また，言葉の定義を疎かにしていたということもある．特に「提案」という言葉に踊らされた．嶋津先生から「提案が欲しい」という言葉を頂いてから，なにか既存の機能に加え，新たな機能を実装しなければならないと考えてしまった．この問題はかなり長引き，設計にも支障を来たすため，数週間に渡って完全にプロジェクトがストップしてしまっていた．提案が「相手の要求を引き出すもの」という言葉のイメージをしていたが，それだけではなく，相手が喜びそうな機能を実装することだけでもいいし，成功報酬に関しても提案した事項であると言える．1 つの言葉を具体的な 1 つのイメージで捉えて捕らえてしまったことが問題の原因であり，これに対してのチーム内での討論が足りなかったこと，及びプロジェクト活動の経験の少なさが勘違いを生んでしまったと考えられる．

さらに，ヒアリングやミーティングの目的を明確にせずに臨んでしまったということも挙げられる．「今日はプロジェクト提案書に承認を貰うまで帰らない」といったように，そのヒアリングで何を達成したいのかと言うことが明確になっていなかった．そのため，ただプロジェクト提案書を提示し，満足できない箇所を指摘され，次回までの修正を約束する，という流れを何度も繰り返してしまった．これは，ヒアリングの相手に，このヒアリングが何のヒアリングなのかも伝わっておらず，お互いにとって満足のいく話し合いができていなかった．この原因はクライアントとのコミュニケーション不足であると考え，12 月 7 日に行われたチーム内ミーティングで解決法を考えた．その方法として，先述したように 12 月 13 日の遠峰様とのミーティングに臨む際に「ミーティング用レジュメ」というものを作成した．この「ミーティング用レジュメ」は，その日に話し合う内容が記されているという実に単純なものであるが，これを作成することにより劇的に話し合いができるようになった．効果として，まずミーティングに臨

む前に何を決める必要があるのかということを話し合い，チーム内で意思を統一し，決定できる．ヒアリングやミーティングの相手に対しても，今日は何を話すのか，何を決めるのかということが具体的に伝わるため，話し合いの結論が出るまで討論をすることができるようになる．例えば 12 月 13 日のミーティングである．プロジェクト提案書の合意を貰うことが目的であったが，満足の出来ない箇所があるのならば，そこを修正して提出すれば合意を貰うことはできるのかという事，また成功報酬などの満足の出来ない箇所はどのように修正するのが良いかを一緒に考えることができ，それまで長い間がかかったプロジェクト提案書の合意をその日に貰えることができた．初歩的なことであり，必要性に気付くのが遅かったということは反省点ではあるが，何事も目的を明確にし，それを相手に伝えることが大事なのだということを学んだ．

2. 実現性調査

1. 実施期間

   2006 年 10 月 19 日～2006 年 10 月 26 日

2. 作業経緯

   実現性調査は以下の 2 点において行った．

   1. GeocodingAPI への接続
   2. DB への接続

   この実現性調査の目的として，実際にプログラムを作成するにあたり障害となりそうな箇所の問題を予め解決しておくことにある．

   「GeocodingAPI への接続」は Web 上の緯度経度算出サービスへの問い合わせの方法の確認である．

   「DB への接続」は，データベースを扱うことが出来るかということのテストであるが，実際には JDBC 経由で Java のプログラムからデータベースを参照，挿入をすることが出来るかということが調査の概要である．結果は使用可能ということで調査を終了した．

   使用するデータベースに関してであるが，プロジェクト発足当時は，本番環境で使用されることとなっていたデータベースは IBM の DB2 とのことだったが，クライアントの意向で急遽 PostgreSQL を使用することとなった．実現性調査では Microsoft 社の Access をデータベースとしてとして扱っていたが，Java プログラムから JDBC 経由で接続するということの実験なので，データベースの種類というのは大きな影響はなかった．実際にクライアントの指定してきた PosgreSQL というのも，あくまでシステ

ムを構築するのに標準的なデータベースなのではないのかという配慮からであり，これでなければならないというものではない．最終的に本番環境で使用されることとなったデータベースは MySQL である．

「GeocodingAPI への接続」は佐藤が，「DB への接続」は松田がそれぞれ担当し，期間内にテストプログラムを作成し，チーム内に配布した．

3. 反省・感想

実現性調査において反省点は見当たらず，有効な活動ができたと言える．実際の実装時においての不安な点は解消され，実装効率の面から見ても実現性調査のテストプログラムのソースコードを使いまわせば良く，余計な手間が省けたということもある．機能を実装する手法が具体的になっているため，プログラムの設計及びテストの設計にも役に立った．

3. 詳細設計

1. 実施期間

2006 年 10 月 19 日～2007 年 12 月 14 日

2. 作業経緯

開発しようとしているシステムが実現可能なものであるという確信を実現性調査でつかんだ後，詳細設計段階に移行した．

詳細設計の目的は，ここで作成した設計書を元にプログラムの仕組みが把握できるようにすることである．そうすれば作成時に設計書を元にプログラムのイメージをすることができるし，実装の担当を決めることができる．

このプロジェクト段階は平澤を中心に行われた．実際に使ってもらえるものを構築するため，システムの管理者が管理しやすい設計にする必要があった．最初にシステムの処理概要を HCP チャートやシーケンス図に整理することからはじめた．特に通常及び例外処理を記載した HCP チャートは丁寧に作成し，重点的に評価した．

最良品質のものを目指し，システムの設計はチーム内でのレビューを重ね，何度も修正された．また地名が改正された時の事態に備え，データベースのテーブルも工夫して設計する必要があった．システムの柔軟性も考え，重要なシステム設定は外部の設定ファイルで調整できるようにし，動作履歴をログとして残しユーザが管理しやすい設計にした．

3. 反省・感想

詳細設計期間は予定より 6 週間ほど遅れて終了した．その大きな原因として，詳細設計をクライアントに承認していただけるものだと勘違いしていたところにある．こちらがどのように設計しても，クライアントが要求した仕様を全て満たしていればよかった．しかし，そのことはプロジェクトが半分過ぎた時点で気が付いた．おまけに，新しく提案した内容もいちいちクライアントに確認させようとしていた．よって，詳細設計もその分延びてしまった．その上，メンバーのネットワーク関連の知識が浅いままで一度設計したこと反省すべき点である．その結果作り上げられた，プログラム仕様書は何度も修正することになり，実装段階でもう一度修正することになった．1 月 19 日の松澤さんのレビューで初めて専門家のアドバイスを詳細設計に反映した．

4. テスト設計

1. 実施期間

   2006 年 10 月 5 日～2007 年 1 月 20 日

2. 作業経緯

   テストの設計は，どのようなテスト項目が必要かをピックアップし，穴の無いテストを構築する手助けとすることを目標とした．ここで作成するテスト設計書はクライアントの承認が必要であり，設計書に問題が無いと取られたらそれを元にテストプログラムの作成を行うことになっていた．これはシステムの実装よりも先にテストプログラムを作成するというテストファースト手法でプロジェクトを進めるための方針である．

   テストの設計は松田がチーフとして行った．プログラム仕様書を元に，テスト仕様の洗い出しをし，それを元にテストケースを記述してゆくというものであった．

   しかし，途中プロジェクト全体のスケジュールが遅れてしまったということもあり，テストの承認を貰うことよりも実装を行うことを優先してしまった．そのため，テスト設計書の承認を貰えたのは 1 月 15 日と実装が完了された段階になってからとなってしまい，テストファーストで行うことができなかった．

   テスト仕様書の中身であるが，プログラムの設計が変わるたびにテストの仕様も変わっていった．これもテスト設計書の承認が遅くなった要因の一つである．一旦は 12 月 15 日に完了したが，その後プログラムの設計が変わり，最終的には 1 月 11 日に完成し，その成果物を 1 月 15 日に承認を貰った．

   さらに，その後，1 月 19 日に行われた松澤さんによるソースコードレビューにより，プログラムの処理が変わったこれを受け，テストの内容にも変化があったために設計書に更新が必要となった．この際は，先にテスト仕様を考え，テストプログラムを作成

してからシステム本体を書くという手法で行ったため，テストファーストでできた．この際に使用したテスト仕様を記載したものが最終的なテスト設計書として，1 月 20 日に完成した．これに関しては遠峰様から承認を得ていないが，12 月 25 日に口頭で変更があったことは伝えた際には特に問題無いと言われた．

3. 反省・感想

何より，テスト設計を実装前に提出しなかったことが反省点である．確かにすぐにでも実装に移りたかった気持ちはあったであろうが，作業の効率を考えると間違いなくテストファーストで行ったほうが速い．現に，1 月 19 日の松澤さんからのレビューがあった後はテストファーストで行い，修正が必要な点を 1 日で実装し直すことができた．このことから考えて，明らかに順序が間違っていたと言える．プログラムの仕様を確認するテストの仕様が曖昧であるのにプログラムが動いているかどうかの確認などができるはずもない．また，プロジェクト開始時にクライアントと交わした「テストファーストで行う」という約束にも反することであり，十分に反省が必要な点である．

5. 実現性調査２

1. 実施期間

2006 年 12 月 13 日

2. 作業経緯

実現性調査 2 は，当初は行う予定は無かった．しかし，途中に要求仕様の変更によりソケット通信の実装が必要となったため，実装にかかる負担を減らすべくその調査を行った．

佐藤が担当した．この際，要件定義のせいでプロジェクトがストップしまっていたということもあり，使用が決まっていた MySQL への接続や以前の実現性調査の成果物との連携といった，出来る限りの仕様の実装を行った．最終的に，設定ファイルとキャッシュのクリア期間以外の部分を実装したものが完成した．

3. 反省・感想

簡単な調査内容であるということから，それまでに調査した内容の実装を行い，実装前の仕様確認時にデモとして使用できるレベルのものができた．実装が 1 日で完了したこともあり，実装に必要な技術が高度なものではないということ，十分期間内に実装

可能であるということがわかった．また，実装に流用可能なコードが作成できたということもあり，この後の活動において非常に有用な情報を得ることができたと言える．さらに，12 月 13 日に行った遠峰様とのミーティングにおいて，実装に先立ってのデモとして見せることができた．この際には遠峰様から「もう殆どできている」という言葉を頂くことができ，安心感を与えることもできた．

6. 実装

1. 実施期間

   2006 年 12 月 14 日～2007 年 1 月 26 日

2. 作業経緯

   実装段階は平澤がメンバーの実装担当を決めることから始まった．実装内容が振り分けられた後，各メンバーは任された箇所を実装するという方針で進んだ．最初は次のような担当で作業行った．

   佐藤： Geocoding API 関連，緯度経度情報キャッシュテーブル関連

   平澤： 設定ファイル関連，ソケット通信関連

   松田： ログ出力関連

   ソースコードの共有及び管理は CVS を通して行った．一通り実装がすんだ後，各メンバーは担当箇所のテストプログラムを作成しシステムの動作確認をいった．各自自分のマシーンで動作検証を行ったあと，ソースコードの結合を行い，納品に備えて本番環境構築を行った．しかし納品直前に行われた松澤さんのレビューの中で，プログラム設計に重大な誤りがあることを指摘され，単体テストが通らない箇所が発見された．おまけに，ソースコードが重複している部分が多数発見された．結果，テストプログラムとプログラムの骨格部分である緯度経度情報を返す部分をリファクタリングしなければならなかった．

   リファクタリング作業は 1 月 22 日に完了した．このとき作業負荷が均等になるように各箇所の担当者を振り分け，テストプログラムから先に実装した．短期間で終える必要性があったが，しっかりした段取りでリファクタリング作業を完了することが出来た．なお，各箇所の修正者は次のように振り分けられた．

   佐藤： データベース及び Geocoding API のやり取りを管理する部分

   平澤： データベース関連，ログ出力関連

   松田： Geocoding API 関連

3. 反省・感想

実装期間が予想以上に長引いてしまった原因の一つとして実装環境構築に関する問題を放置したためである．メンバーのうち松田だけが Mac を使用していたため，元々使用するはずだった Postgres のインストールに手間取り，急遽データベースを MySQL に変更する必要があった．更にソースを管理する CVS にアクセスできない問題もあり，それは 1 月 19 日にようやく解決された．

もう一つの原因は実装担当の振り分け方にある．最初はメンバーの負荷をあまり考えずに振り分けたため，作業負荷が重点的にメンバー一人に掛かってしまった．おまけにクライアントを早く安心させたいという気持ちもあり，テストモジュールの作成を後回しにした．結果単体テストが全て通らない状態で，クライアントに納品しようとしていた．これは大いに反省すべき点である．

7. 単体テスト

１．実施期間

2006 年 10 月 5 日ー2007 年 1 月 25 日

２．作業経緯

GeocodingModule のコーディングが終了した後，クライアントに納品するにあたって必要な仕様を満たしており，それが正常に動作をしているかどうかをテストした．単体テストでは JUNIT を使用することでテストを行う．

納品日は同日２００７年１月２５日であったため，急なテストだった．今回製作した GeocodingModule はバックグラウンドで動くサービスであるため，表面的にそのモジュールを評価することは難しい．そのため，確実にサービスを実行できるということを重要視し，g-mod プロジェクトでは「品質第一」ということを重要だと考えている．それを実現するために早い段階からテストケースの作成が行われていた．

２００６年の１１月から１２月にかけてテスト設計書の製作は行われていたが，最終的にはプログラム設計の変更などもあり，１月中旬頃にテスト設計が完成した．納品の前にテストを行いエラーを出さなかったが，一つ一つのケース毎でテストを行ってしまい，全体でテストを行うことがなかった．また古いテストケースを使用していたことや，テストの個数が足りなかったので最終的にテストが終了したのは納品後になった．結果はテスト報告書に記載してある．

３．反省

JUnit でテストを作成しているにも関わらず，そのテストの設計に不備があり，結局のところ納品後にテスト設計を再度やり

直すこととなってしまった．作業経緯でも述べたように一つずつケースをテストするということをしてしまい，全体を通してのテストということを行わなかったという反省点が挙げられる．また同様に古いテストを行っていたことや，テスト数が足りなかった点も挙げられる．

8. 結合テスト

1．実施期間

2006 年 10 月 5 日ー2007 年 1 月 25 日

2．作業経緯

単体テストと同様に，作成したシステムが正常に動いているかを確認することを目的とし，システム全体のテストを行った．

テストはテストシナリオに沿って行った．テストシナリオに基づき正規フロー，異常フローに分類し，それぞれ正しい動作をしているかということをテストする．行ったのは納品直前である．

3．反省

結合テストを行ったのが納品直前となってしまった．

9. クライアント検証

1. 実施期間

2007 年 1 月 25 日以降

2. 作業経緯

クライアント検証は，納品する物が正しいかをクライアントに判断してもらうことである．G-mod チームとしては，本番環境下にあるプログラムのデモを行い，その後電子媒体で納品物を送信し数日中に返事をもらう，という方針で行った．

1 月 25 日，遠峰様に大岩研究室まで来てもらい，g-mod プロジェクトメンバーによりデモを行った．ここで行ったのはあくまでデモであり，擬似クライアントを使って機能が仕様を満たし動いているかどうかということを確認してもらうためのものである．

一度遠峰様には DMC に GeocodingModule を持ち帰ってもらい，検証をしてもらうこととなった．

それで検証用の提出が終わる予定であったが，エンコードに関して気になる部分があるのでチェックして欲しいとの要望があり，本番環境下のプログラムを多少いじってしまった．それが直接的な原因ではなかったものの，突然仕様どおりに動かなくなってしまった．

その後，プログラムを修正し，再納品したという件で遠峰様にメールをし，これにて提出を完了した．

その後，1 月 30 日に DMC の遠峰様に仕様を満たしているというのを確認していただき，検証を終了した．

3. 反省・感想

一旦検証用に差し出したものを編集してしまったという，通常考えられないことをしてしまったことが最大の反省点である．編集した点が直接の不具合ではなかったが，一旦見せたものを改造するという考えられないことをしてしまった．二度としてはいけないことであるし，十分に反省しなければならない．

ここでミスをしたおかげで見つかった不具合を修正することができたが，せめてクライアントに断りを入れ，別のファイルとして保存するべきであった．

10. ドキュメント作成

1. 実施期間

2006 年 10 月 5 日～2007 年 2 月 23 日

2. 作業経緯

ドキュメント作成の目的は，本プロジェクトの当初にクライアントからの要求を元に考案した必須ドキュメントの記述を行うことである．

他のプロジェクト段階と同じく，ドキュメント作成期間もスケジュールより遅れて行われた．今期のプロジェクトでは全てのドキュメントに更新履歴を記載し，変更箇所をすぐ確認できるようにドキュメントを書いた．特にプロジェクトの意思表示であるプロジェクト提案書とプログラム仕様書度もレビューを通して修正された．全てのドキュメントが完了したのは納品当日でなり，予定より一週間ほど遅れた．

導入手順書

3. 反省・感想

他のプロジェクト段階が長引いたこともあり，ドキュメント作成は他の段階と同時並行で行われることが多かった．これはプロジェクト提案書に書かれた内容に対して，1 月になるまでクライアントの合意が得られなかったことが大きな原因である．結果，常にドキュメントを書いているというあまり望ましくないプロジェクトの進め方になってしまった．

ドキュメント作成に関してもう一つ反省すべき点がある．プロジェクト開始時にドキュメントにそれぞれ番号をつけて管理することに決めたにも関わらず，ほとんどのドキュメントに番号は付いていない．「管理しやすいからそうしよう」という事で決めたにも関わらず，面倒がったり，そのルールを忘れたりして結局無視してしまい，そのままプロジェクトを進めてしまったことも大いに反省しなければならない．先に苦しめば後から楽になるはずなのである．

11. 本番環境リリース

1. 実施期間

2006 年 12 月 18 日～2007 年 1 月 25 日

2. 作業経緯

本番環境リリースの目的は，納品をするためではなく，そのためのクライアント検証が行える状態に持っていくことである．

本番環境においてのリリースであるが，12 月 18 日の遠峰様とのミーティングにおいて，用意して頂きたい本番環境についての話し合いを行った．この際に決まったのは，OS は細かい設定が必要なため遠峰様が用意するが，その他のものは全て g-mod チームが入れるということである．そのため，必要なスペックをチーム内で検討し，遠峰様にお知らせするということになった．

スペックは，OS は Debian GNU/Linux，メモリは 256MB，HDD は 7GB でお願いした．

その後，12 月 21 日，遠峰様を交えて本番環境の整備を始めた．この日の目標としては Java と MySQL のインストールが完了することであり，あとはシステム本体を入れればいいだけの状況に持っていくことであった．apt-get コマンドで簡単にできると想定していたが，インストール可能ソフトウェア一覧にはバージョンが古いものしか無かったため，公式ホームページからダウンロードするところから始めなければならなかった．この日は遠峰様の体調が良くなかったこともあり，未完了のまま終わってしまった．この続きは，外部からアクセスして環境整備ができるよう，ホストのアドレスを追って知らせるとの言葉を遠峰様から頂いたが，年明けの 1 月 16 日まで頂くことができなかった．

結局本番環境整備は，実装も完了した後，1 月 23 日に行うこととなった．この際は大岩研究室に所属する先輩の助けもあり，その日に環境を整備することができた．

システム本体の導入は 1 月 24 日に行った．この際，必要となるシステムコマンドファイルについての調査やパスの通し方等の障害もあったが，無事導入を終え，実際に動いているところを確認することができた．

1 月 25 日，遠峰様に成果物のデモとして使用したのもこの環境下のものであった．しっかり動いていることを遠峰様が確認し，これで本番環境リリースとなった．その後，バグが発覚し修正を行ったが，修正に関しての連絡を入れ，仕様通り動いていることを確認した．

3. 反省・感想

本番環境リリースの日が遅れたのはプログラムの実装によるものが大きいが，「あとはプログラムを入れるだけ」という状態に持っていくことはもっと早くできたであろうし，何も納品の 2 日前に行う必要は無かった．

遅れた要因として，遠峰さんからの連絡をひたすら待ちすぎたということが挙げられる．今プロジェクトにおける活動全般に言える反省点であるが，回答の期限を約束することを怠ったことは反省点である．もっと早く整備したければ，遠峰様に連絡を入れるべきであった．

12. 納品

1. 実施期間

2007 年 1 月 25 日

2. 作業経緯

納品は，提出すべきものが全てクライアントにとって望ましい状態で提出されたかを判断するために行った．本プロジェクトにおいては，クライアントによる検証の完了を納品とした．

3. 反省・感想

上記と同じであるが，明確な回答の期限を頂かなかったことにより，最終報告書の記述に支障を来すということがあった．プロジェクトを通じて最も多く犯したミスである．

13. 中間報告

1. 実施期間

   2006 年 11 月 9 日

2. 作業経緯

   大岩研究室のプロジェクト中間報告．g-mod プロジェクトでは DMC システムと GeocodingModule との関連性について終始説明をした．これまでの g-mod プロジェクトの進捗報告では「何をしているのかがわからない」という意見が多かったため，今回の中間報告ではそれを払拭するのが狙いだった．発表では時間を目一杯使い詳細な説明を用意した．

3. 反省・感想

   この中間報告での反省点は自分たちが何をすべきかということを，この時点で初めて気づいたということである．中間報告では GeocodingModule がどういうものかということを説明すべきだったにも関わらず，あくまで DMC システムの概要を DMC に替わり代弁していたに過ぎなかった．しかし，この中間報告のあとでもまだプロジェクトの目的と，目標をしっかり把握することが出来ていなかった．目的，目標を理解することになったのは中間報告後に DMC の嶋津様とのヒアリングを行った以降ということになる．これを受けて g-mod プロジェクトでは DMC の目標と g-mod プロジェクトの目標が別なものであるということを初めて理解した．

X. プロジェクト成果

上記のプロセスを経て，本プロジェクトの活動によってどのようなことを学ぶことができたか，また学習目標の達成をすることができたかどうかを考えていく．

1. 学習結果

本プロジェクトにおいて学んだことは以下の通りである．

1. プロジェクトの目的の定義

   本プロジェクトでは，プロジェクトの目的及び目標を決めないままプロジェクトを進めるということをしてしまった．それにより，このプロジェクトは何を達成しようとするのか，何を持って成功とするのかということを考えないままプロジェクトを進めた．これによる影響として，自分たちが DMC 様の一体となっていると錯覚してしまったことが挙げられる．そのせいで，DMC 様から言われることは守らねばならないとか，自分たちは DMC 様に言われることをやればいいんだと勘違いをしてしまった．これが要件定義の際に指摘されたことであり，プロジェクトの進行を最も遅らせた要因である．

   プロジェクトを開始する際に目的・目標を決めることの意味，そして重大さを学ぶことができた．

2. 交渉

   このプロジェクトに参加するメンバーは殆どクライアントからの明確な要求を持ってのプロジェクトに参加した経験が無かったため，交渉を行うことに対する経験も多くなかった．そのため，話し合いが交渉にならず一方的な要求で終わったり，また交渉内容のポイントがずれていたりするということがあった．

   交渉にならなかった原因として，上記のこととも関連するが，クライアントには従わなくてはならないという思いがメンバー内にあり，提案事項を却下されたり，新たに要求されたことがあったりしたとき，そのままそうしなくてはならないと思ってしまい，話し合いを終了させてしまった．ただ言いなりになっているだけであり，交渉をすることができなかった．

   交渉のポイントがずれていたことの例として，ある約束したときに期限の日にちを決めることをしなかったということや，承認

を貰うべく事項の承認を貰わないまま話し合いを終了させてしまったということが挙げられる．この原因として，何を話し合うべきなのか，何を決定すべきなのかということを考えずにただ話し合わなければならないという使命感に任せて交渉を行ってしまったという事と，決定した後に何をいつまでにする必要があるのかといったプロジェクト全体を見るビジョンが欠けていたことが挙げられる．

クライアントとの話し合いに限らず，話し合いの目的は何なのかということを，「交渉」をする内容と方法を学んだ．

3. DB の設計手法

本プロジェクトでは，DB の設計で 2 度躓いたことがある．1 度は実装前の設計段階で，DB のテーブル設計においてであり，もう一度はクライアント検収後に不具合が出た際の DB 再設計である．

一度目の躓きから見ていく．当初は，DB はテーブル 1 つだけで十分という設計で行った．しかし，扱う必要のあるデータが増えると共にテーブル 1 つでは足りないと気付く．その際，複数のテーブルに分割したのだが，どのように分割するかで問題となった．このとき，各テーブルがどのような役割を持っているのかを考え最終的なテーブルが設計されたのだが，目的によってテーブルを変えるという考えは当初は無かった．つまり，設計の目的を見失っていたということである．設計を行うときに，何を設計しその目的は何かということを考える必要の重要性を再確認した．

二度目の躓きであるが，これは主キーの設定ができていなかったことが原因で不具合が起こったことである．600 以上のデータが入っている 3 つのテーブルの中から特定の組み合わせを持つデータを取得しようとしていたのだが，あまりに時間がかかってしまったため，タイムアウトを起こしてしまっていたのだ．この原因を探るために使用したのが「EXPLAIN」コマンドである．これは「EXPLAIN SELECT～」のように使うと，どのようにそのコマンドを実行しているかがわかる．そこで「rows」のカラムを見ると，全データをくっつけた中からデータを探そうとしており，実に 600 の 3 乗もの組み合わせの中から検索しようとしていた．主キーを設定すると，そのキーは検索キーとして他のキーよりも先に見つかるため，該当データが 1 つの場合は 1 つずつのデータを抽出したものをくっつけることとなり，わずか 1 通りの組み合わせで済むわけである．単純なことであるが，多大な効果があることがわかった．

4. 実行ファイルの作成

実行ファイルの作成であるが，UNIX 上でどのように実行させるファイルを作るかを知らなく，シェルスクリプトについて調べるところから始めた．記述内容であるが，単純にコマンドを指定させれば良いだけであったが，なかなか上手く通らなかった．この原因として，パスの問題と命令文のミスがあった．パスの問題であるが，パスが通っていないファイルを実行しようとしてもそ

のフォルダ下になければ実行することができないため，パスを指定する必要があった．命令文のミスとして，ウィンドウズ環境では「;」でよかった箇所が UNIX 上では「:」でなければならないということがあった．

2. 学習目標

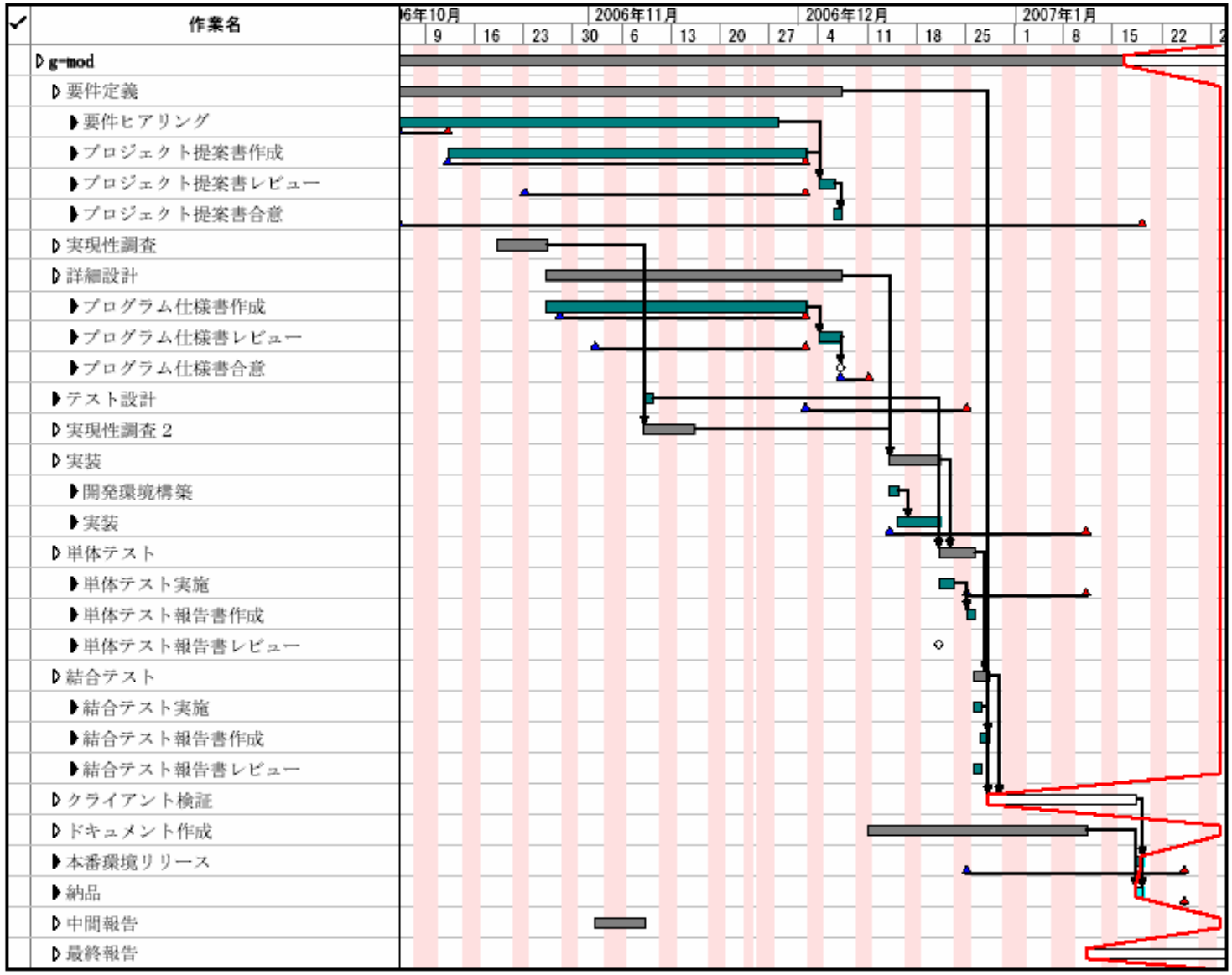
本プロジェクトにおける学習目標は「新しい事への挑戦」であり，プロジェクトを通して，今まで行っていなかったことや苦手だったことに積極的にチャレンジして学んでいくことを心掛けた．

結果であるが，上記のような事柄を学ぶことができたため，学習目標は達成できたと評価する．ただ，それが積極的である証拠は無く，幾つ勉強すればといった数も指定していなかったため，完全に自己評価になってしまい，あまり意味を持たない目標となってしまった．ここにも，目標を定める際は慎重に行わなければならないという反省点を当てはめることができる．

XI. プロジェクト実績

1. スケジュール

スケジュール通りに進めることが出来なかった.
要件定義の遅れを引きずったまま最後まで進んでしまった影響もあり，当初予定していたクライアント検証を，クライアント側で作成するプログラムが完成していないことや納品が遅れてしまったこともあり，実施できなかった.
WBS で洗い出せていないタスクもあり，当初のスケジュールから大きく変更が入ってしまった.
WBS 作成時点でプロジェクトの進捗イメージが出来ていなかった事が原因と感じる.
学生と会う機会が１週間に１度しか会えない状態であることから、会ったときに“どんな作業をするのか”、“いつまでに行うのか”を明確に指示する必要があり、ＷＢＳのレベルをもっと細かいレベル（３段階か４段階）まで落とす必要があったように思われる.
また、クライアントとの打ち合わせスケジュールが考慮されていなかったことも反省点の一つとなりコミュニケーション計画が欠落していたこともスケジュール遅延の大きな要因となっている.
前期の“Soundonly”や今期の“４hands”はタスクが細分化されており、毎週的確な指示ができていたことでプロジェクトをスムーズに進行させることができていたように感じる.
洗い出せていなかったタスクの１つであった「実現性調査」はクライアントの信頼を得るのに非常に効果があった.
「実現性調査」で作成したプログラムをクライアントに見せることで信頼を得ることができたからである.
また、実装フェーズに入った際も技術的に心配なことや悩む事なく作業を進めることができた.



2. プロジェクト規模

1) ステップ数

GeocodingModule 実行ステップ数：1,105 ステップ

プロジェクト計画時点でステップ数の見積は無いので実績のみの記載となる.

今までの経験から各クラスの機能に対して実行行数は妥当な数と感じる.

No.	ファイル名	Type	実行	空行	コメント	合計
1	Geocode/APIManager.java	Java	**196**	29	101	326
2	Geocode/GeoDataManager.java	Java	**77**	11	38	126
3	Geocode/RequestProcessor.java	Java	**41**	11	21	73
4	Geocode/DBManager.java	Java	**319**	33	105	457
5	Geocode/GeoData.java	Java	**34**	15	21	70
6	Geocode/Geocoder.java	Java	**57**	14	47	118
7	log/SystemLogger.java	Java	**23**	22	6	51
8	log/LogWriter.java	Java	**90**	26	93	209
9	log/AccessLogger.java	Java	**11**	11	6	28
10	main/GeocodingModule.java	Java	**78**	16	32	126
11	setting/GeocodeSettings.java	Java	**179**	50	82	311
	合計		**1105**	238	552	1895

2) 作業工数

要件定義の工数が大幅に違ってしまった.

これは初期段階におけるクライアントへのヒアリング不足が大きな原因となっている.

途中メール不調によるクライアントとのすれ違いが発生してしまった事も原因となっており、要件が確定しないままズルズルとプロジェクトが進んでしまった。　その結果、後に控えている全ての工程に手をつける事が出来ず遅延が発生した.

詳細設計では要件の変更は無いものの，詳細設計及びプログラムの動作に関わる変更が発生した為に詳細設計書の修正が入る回数が多かった.

No.	工程	見積工数(h)	実績工数(h)	工数差分（実績-見積）
1	要件定義	40	124.5	84.5
2	実現性調査	20	45	25
3	詳細設計	40	103	63
4	テスト設計	40	52	12
5	実現性調査2	20	13	-7
6	実装	144	107.5	-36.5
7	単体テスト	16	57	41
8	結合テスト	24	21	-3
9	ドキュメント作成	40	7	-33
10	本番環境リリース	16	15	-1
	合計(人時)	400	545	145
	合計(人日)	50	68	18

3. 品質

1) レビュー

レビュー参加者は本人以外に 2 人で行う.
作成時間の１０％はレビューにあてる.

ソースコードレビューは松澤さんによる十分なレビューを受けたが，その他ドキュメント類においては
作成時間の 10%のレビュー時間を満たしていなかった.
レビューは対面形式で行うことを想定していたが，メールでのレビューが中心となり意思を伝えきれないこともあった.

2) 規約

規約に準拠したコーディングを行う.

なるべく規約に沿ったコーディングを行うように心掛けるということで，テストツールを用いてコード解析を実施した.
※添付ファイル参照

3) テスト

テスト数は 100 項目以上.
バグ摘出指標　50 個/Kstep.

1 度テスト項目を消化したが，NG 件数が 0 だった為に項目の見直し及び再テストを実施した.
単体テスト結果
・テスト項目数：５８
・○：４２
・×：１６
項目数及び摘出×数が少ないが，これはコーディングを行いながら実行した為である.

4) レスポンス

基準値５秒/レスポンス.

レスポンスはシングルタスクで実行して 1 秒程度となる.
品質基準 5 秒はクリアされている.

4. 総合評価

プロジェクト計画からは大きな差異が発生してしまった.
特にプロジェクト初期段階（要件定義）での遅れやプロジェクト目標である品質の定義においてチーム内で認識を統一することで多くの時間を費やしてしまった.　プロジェクト初期段階での遅れを取り戻すことができないまま最後まで進んでしまい，中間報告以降はやり残している上流工程を行いながら並行して下流工程を行った為作業時間も多くなってしまった.
本プロジェクトでは品質＝顧客満足度を目標にしており、作成したプログラムはクライアントの満足できるものになったと思われる.
しかし、プロジェクト経緯からもわかるように、プロジェクトの進め方については様々な反省点が挙げられる.
クライアントに不信感を与えてしまったことや、プロジェクトの目的・目標を見失ってしまったこと、工数見積とのブレが大きかったことなどである.
最終的に納品することが出来たが、プロジェクトとしては失敗してもおかしくなかった。
プロジェクトを支えてくれたのはメンバーの前向きな信念と努力によるものである.
非常にプレッシャーの強いプロジェクトであったが、メンバーにとって多くの学習成果があったことやなにより達成感を感じることが出来たことは非常に喜ばしいことである.

以上.