修 士 論 文

短期的な協調型開発の失敗要因調査に基づくハッカソン支援システムの研究開発

北陸先端科学技術大学院大学
知識科学研究科 知識科学専攻

西 康太郎

2015 年 3 月

修 士 論 文

短期的な協調型開発の失敗要因調査に基づくハッカソン支援システムの研究開発

指導教員 西本 一志 教授

北陸先端科学技術大学院大学
知識科学研究科 知識科学専攻

1350029 西 康太郎

審査委員： 西本 一志 教授（主査）
内平 直志 教授
Dam Hieu Chi 准教授
Ho Tu Bao 教授

提出年月: 2015 年 2 月

A Hackathon Support System based on Failure Factor Investigation in Short-term Collaborative Product Development

Kohtaro Nishi

School of Knowledge Science,
Japand Advanced Institute of Science and Technology

March, 2015

Keywords: Hackathon, Death March, Prototyping, Groupware, Group Awareness, Collaborative Action.

This paper proposes a support system of a group hackathon named “HackathonMediator” to solve problems of collaboration in a team of hackathon.

HackathonMediator consists of two functions: a function to share work screen among members for sharing states of progress and a function to create a mock-up model for sharing product images.

We applied this system to an actual Hackathon event. Results of evaluation by behavioral observations and interviews are reported.

目次

表目次

図目次

第 1 章

はじめに

1.1　背景

1.1.1　ソフトウェア開発現場の進歩と変化

近年，IT 産業は幅広く開拓され，ソフトウェアの流通およびその開発形態は共に多様化・高速化を遂げつつある．多くの開発リソースが誰にでも再利用可能な形で配布されるようになり，Web アプリケーションやスマートフォンアプリケーション界隈を筆頭として，ソフトウェア市場に対する個人規模での参入が容易になった．プロダクトの開発リソースは API（Application Programming Interface）や Framework などといった形で提供され，新たな開発者はこれらを活用することにより，プロダクト開発のコストを大幅に削減することが可能である．開発リソースの配布により，経験の浅い開発者に対しても技術的なハードルが緩和され，多様なプロダクトを作り上げることが容易になり，アマチュア開発者がプロダクト開発に挑戦する機会をもたらした．また，開発工数の短縮に繋がったことにより，プロフェッショナルの開発者による業務においてもリリース・レビュー収集などを反復して行う短期的なアプリケーション開発形態が促進された．昨今では，職業・年齢等に関係なく参加可能なコンテスト形式の作品制作会が盛んに開催されており，上記のような要因を伴って，開発現場におけるプロフェッショナルの開発者とアマチュアの開発者によるコラボレーションの機会が増加しつつある．

1.1.2　オープンデータ戦略によるイノベーション促進

近年，産業やサービスに新たなイノベーションを促す取り組みとして，我が国はオープンデータの活用を推進している [1][2]．オープンデータ戦略とは，政府や自治体などが保有す

るデータを無償かつ誰もが二次利用できる形式で公開することにより，企業や個人に対し新たなサービス等を提案できる余地をもたらす取り組みである．また，近年では行政機関のみならず，民間企業においても自社保有のデータを一部公開し，それらの活用を促進するためにコンテストを開催する例も多い [3]．これらの取り組みは，本来消費者としての立場にある人々（＝エンドユーザ）に対し新たな産業・サービスの創造・発展に寄与する機会を与え，オープンデータ提供元のコミュニティに新たな視点をもたらす期待がある．昨今では，データを保有する各団体が作品制作コンテストの主催団体に協力し，参加者にオープンデータの活用を促すなどといった「○○× IT」のコラボレーション事例も増加してきている．節 1.1.1 で述べた作品制作コンテストの増加も伴って，より多くの開発者が組織の枠組みを超えて共同となり，社会的な問題解決に取り組む機会を得るようになった．

1.1.3 ハッカソンについて

プロダクト開発形態の変化を受け，オープンデータ戦略やイノベーション促進活動の一つとして，ハッカソン（Hackathon:Hack-a-thon）という取り組みが現在注目されている．ハッカソンは「Hack」と「Marathon」からなる造語であり，「多くの人々がノンストップで一つの物事に熱中して取り組む」ことを指す．世界初のハッカソン開催事例は 1999 年に OpenBSD がカナダのアルバータ州で実施したイベントであるとされ [4]，2011 年頃より我が国においても導入事例が増加し，2015 年 1 月に至るまで我が国のハッカソン認知度は上昇の傾向を示している．ハッカソンでは，運営団体が設営する会場内に多数の参加者が集い，3-6 名程度のグループ（多くの場合が即席である）を結成し，1-2 日ほどの短期間において革新的な成果物を生み出すことが目標とされる [5]．多くのハッカソンにはアイデアの発散と収束を伴う企画段階があり，開発段階にてそのプロトタイプを実現し，最終的に聴衆の前で公開プレゼンテーション（多くの場合はデモンストレーションを含む）を行う．昨今においては IT 企業がコンテスト形式のハッカソンを主催する事例が増加し，新たなサービスや人材を発掘する場としても注目されつつある．

1.2 ハッカソンチームにおける情報共有の問題

ハッカソンには，従来の共同開発のスタイルとは大きく異なる特徴がある．第一に，チームに明確な作業プロセス等の制約が提供されない場合が多い点である [2]．各チームメンバには自発的なマネジメント・アクションの選択が求められる．第二に，従来は回避すべき状態であったデスマーチが前提となる点である．多くのハッカソンにおいて，チームは 1〜2 日程度のハードなスケジュールの中で優れたアイデアを企画し，開発・プレゼンテーション

を行い，革新的な成果物を生み出さなければならない．第三に，現場で即席に構成されるチームによる，期間限定での密度の高いコラボレーションが求められる点である．しかもチームには，プロフェッショナルやアマチュア，プログラマやデザイナなど，様々な技能や技量を持つ人が入り混じる．彼らの日常業務における常識は，ハッカソンチームにおいて通用しない．

ハッカソンはこのような特徴を持つため，成果物を実現するに至らないケースや，チームが途中で空中分解してしまうようなケースなど，失敗に終わる事例が多数見受けられる．今後ハッカソンがさらなる発展を遂げ，参加者と作品としての実績を増加させるためには，このような失敗を回避する手段を実現することが望まれる．そこで本研究では，チーム内での情報共有が成功のための重要な鍵となると考えた．

ハッカソンの最中においてはタスク設定や軌道修正が高速に繰り返されるとともに，情報共有頻度および情報の具体性が徐々に低下していく傾向が見受けられている．特に開発過程の後半にかけて，各メンバは個人のタスクに没頭せざるを得なくなり，チームのコミュニケーション量は著しい低下を見せる．

1.2.1 ハッカソンでの情報共有手段

ハッカソンでの情報共有手段としては以下の 3 つが代表的である．

1. ホワイトボードと Post it の活用
   TODO リスト作成をはじめ，ブレインストーミング，制作物のスケッチやスケジュールテーブルの作成など，様々な活用法が存在する．
2. SNS などと連携したチャットルームの活用
   chatwork[6] や slack[7] などの活用が代表的である．Facebook や Twitter などの SNS アカウントを使用してユーザ登録できるため導入が容易であり，グループメンバのみが閲覧可能なチャットルームを作成できるほか，ファイル転送機能を兼ね備えているため，リソース共有ツールとしての利用も可能である．
3. 共有ドキュメントの活用
   hackpad[8] などの活用が代表的である．hackpad ではドキュメントの共同編集機能が提供され，「どのメンバが，どこに記入したか」を確認することができるほか，埋め込みソースコードに対応したシンタックスハイライト機能などが提供される．2 と同様，SNS アカウントを使用してのユーザ登録が可能である．

これらの情報共有手段は，任意のドキュメントを作成し共有することで情報共有のパフォーマンスを高める傾向にある．これらのツールはユーザに対して受動的であり，チームメイトがツールを積極的に活用することでグループ内での効用を発揮する．主に企画から開発準備段階の間にて高い頻度で活用され，開発過程にて作業文脈に特化した作業ツール（IDE やイラストツールなど）が活用されるようになるとともに，これらの情報共有ツールの使用頻度は低下する傾向にある．また，これらのツールは汎用性が高い点が長所であるが，ツールの効用がユーザの活用スキルやチームの個性に大きく依存するため，誰もが安定して高い活用メリットを得られるわけではない．

1.3 本研究の目的

本稿では，ハッカソン状況下での協調型開発において，チーム内での情報共有に特化した手段を提供するシステム「HackathonMediator」（意：ハッカソンチームの仲介者）を提案する．本システムを開発する上で検討された要件は以下である．

- デスマーチ環境に特化した情報共有手段を提供すること
- ハッカソンの経験差に依存なく安定した活用メリットを提供すること

HackathonMediator は，「成果物イメージ共有を兼ねたプロトタイプ Ver.0 作成機能」，「作業進捗共有を兼ねた画面共有機能」から構成される．ハッカソンの企画段階において「成果物イメージ共有を兼ねたプロトタイプ Ver.0 作成機能」を活用することにより，各チームメンバの思い描くプロダクト像の差異を吸収し，統一されたプレゼンテーションモデル（モックアップ）をチームは得る．この機能を用いて作成されたモックアップは，実際のプログラムコードとして直接活用できるため，企画段階から開発段階に向けて効率的な移行作業を提供する．開発段階への移行後，「作業進捗共有を兼ねた画面共有機能」を活用することにより，各チームメンバは他メンバの作業状況を把握しつつ作業を進めることができる．この機能では，各メンバが特定の作業（アプリケーションのデバッグ，画像作成，プレゼンテーション資料作成など）を行うタイミングに合わせ，該当メンバの作業画面を大型ディスプレイにミラーリングすることにより，各メンバの作業状況をチームが効率的に確認可能な機会を提供する．

本システムの予備実験および評価実験は，北陸先端科学技術大学院大学のゲーム開発交流サークルが定期的に主催しているハッカソンおよび九州産業大学情報科学部で著者が主催したハッカソン「9389」にて実施した．

1.4　本論文の構成

本章では，本研究の背景としてハッカソンについて紹介し，本研究の目的を述べた．以降においては，第 2 章にて本研究が支援対象とするハッカソンの検討，第 3 章にて関連研究，第 4 章にて予備実験，第 5 章にて提案システム，第 6 章にて実験，第 7 にて考察，第 8 章にて本研究のまとめをそれぞれ述べる．

第 2 章

支援対象の検討

この章では，ハッカソンに関して事前に収集した知識およびそれらに基づいて検討した支援対象のハッカソンについて説明する．

2.1　ハッカソンの開催形態の分類

ハッカソンの開催形態を調査するにあたって，著者は（株）角川アスキー総合研究所にて 2014 年 6 月 6 日に開催されたセミナー「ハッカソン、アイデアソンのやり方　～その魅力とオープンイノベーションとしての可能性～」へ出席し，情報収集を行った．本セミナーにおいて学んだハッカソンの開催形態について以下に示す．

- 「サービス事業開発」型
  サービスや事業を前提としたプロダクトの創出を目的としており，企画においては市場分析やマネタイズなどが視野に入る場合もあり，企業主催で行われることが多く，主な参加者傾向はアントレプレナー，エンジニアである．国内の事例としては（株）ローソンによる「HackaLawson」[3] や Yahoo! Inc. による「The Open Hack Day」[9] などが存在する．
- 「キャリア教育・採用・研修」型
  キャリア教育や研修，採用活動の一部として，オープンな環境で組織内外との人材交流を兼ねて開催される．企業と学校が共催する場合もあり，主な参加者傾向は主催組織に関連を持つエンジニアである．国内の事例としては千葉大学教育学部および（株）GREE の共催による「メディアリテラシー教育演習」[10] や宮城県石巻工業高等学校および複数の団体の共催による「石巻ハッカソン」[11] などが存在する．
- 「特定の課題解決・可能性創造」型

異業種人材が交流をしながら，共通の課題やテーマに向かってプロダクトを生み出すことを目的とする．国内においてもっとも多い開催形態であり，様々な分野に対し IT とのコラボレーションを導入することにより，イノベーションの先駆けとして機能する期待がある．特定の課題やテーマに関係する自治体や企業が共同で開催する傾向があり，主催者やスポンサー団体からオープンデータや API 等が提供される場合が多い．参加者傾向は文系・理系問わず様々であり，エンドユーザとしての役割を持った参加者と技術に精通した参加者がコラボレーションする例も多い．国内の事例としては（株）東京証券取引所による「東証ソーシャルかぶコン」[12] や日本放送協会による「NHK ハッカソン」[13]，青森県による「農業× IT マッチングワークショップ」[14]，（株）パソナテックおよび世界コスプレサミット事務局の共催による「COSPLAYTHON」[15] などがある．

- 「新技術のトライアル」型
  新技術やデバイスが発表された際などに関心のある者同士で活用を試み，可能性の発見に繋げることを目的とする．開発コミュニティが主催することが多く，参加者傾向は様々である．昨今においては Oculus Rift，Swift，iBeacon などの活用をテーマとするハッカソンが開催された．

2.2 支援対象とするハッカソンの特定

本研究がハッカソン支援システムを提案するにあたっては，導入可能性のあるハッカソンのカテゴリを特定する必要があった．ハッカソン運営者および参加者に対し実施したインタビューにおいて，ハッカソンの支援システムを導入することに対する意見を求めたところ，「ハッカソン熟練者は自身になじんだやり方を好んで実践する傾向にあり，システム使用の強制を好まない場合もあるだろう」との意見があったほか，「教育を重視するハッカソンによってはシステムの介入を好まない場合もある」という意見を得た．後者の理由は，「コミュニケーションやスケジューリングの不足がもたらす失敗もハッカソンにおける学びの一つである」という考えによるものである．これらのことを踏まえ，本研究が対象とするハッカソンの条件について，まず最初に以下の内容を考慮した．

1. ハッカソン未経験者の新規参入に積極的であること
2. プロダクトとしての成果物の提案を重視していること

多くのハッカソンにおいては，参加者数の増加を目指す上でハッカソン未経験者の受け入れを推奨しているため，1 の条件についてはほぼすべてのハッカソンが当てはまると考えら

れる．特に学際的な視点を求める傾向の強い「特定の課題解決・可能性創造」型のハッカソンにおいては，自治体の構成員などをはじめ，参加者は IT に関連した職業を持つ者に限定されない場合も多い．このことから，本研究では 2 の条件の有無をハッカソン支援システムの導入可能性の目安とし，節 2.1 で述べたハッカソン開催形態を「成果物重視型」と「教育重視型」のカテゴリに区別した（表 2.1）．

表 2.1　成果物重視型と教育重視型のハッカソン

	ハッカソン形態	特徴
成果物重視型	「サービス事業開発」型	産業としての展望を重視
	「特定の課題解決・可能性創造」型	学際的な共創を重視
	「新技術のトライアル」型	新技術開拓を重視
教育重視型	「キャリア教育・採用・研修」型	現場での学びを重視

- 成果物重視型ハッカソン
  成果物重視型のハッカソンにおいては，新規性・展望性などの「定量的な価値」を内包するプロダクトの誕生が期待される．ハッカソンにて優れたプロダクトが誕生することは主催団体の実績に直接的な影響を及ぼすため，主催者は参加者に協力的な姿勢をとり，チームの進行状況によっては作業に介入する場合もある．このような傾向から，成果物重視型ハッカソンにおいては支援システムの導入可能性が高いと考えられる．なお，この「成果物重視型ハッカソン」に該当するハッカソン運営者に「ハッカソンの支援システムを導入したいと思うか」という質問を実施したところ，「運営側としての負担も軽減できるため，そのようなものがあると助かる」という回答が得られている．
- 教育重視型ハッカソン
  教育重視型のハッカソンにおいては，参加者たちに対しチームマネジメント能力を初めとする「開発現場で通用する実践的なスキルの養成」が期待される．「成果物重視型ハッカソン」との大きな違いは，主催者が参加者たちの自主性を尊重する傾向が強いという点である．主催者は参加者に対し必要以上に介入しない姿勢を取り，ハッカソンチームの成功・失敗などといった結果そのものよりも，協調型開発がもたらす経験知の獲得を重要視する傾向にある．このことから，教育重視型ハッカソンにおいては支援システムの導入可能性が低いと考えられる．

ハッカソン開催形態のうち過半数が成果物重視型に当てはまる傾向が強く，審査員（多くの場合はテーマに関連する専門家である）を招いた上で成果物の新規性や利便性・展望性などを評価し，優れた成果物に賞を与える傾向にあるほか，ハッカソン後も開発した成果物に拡張を加えたのちにリリースを目指す動きも多い．

2.3　ハッカソンにおけるチームの成功について

ハッカソンチーム内においてどのような状態が成功と捉えられるかについて，ハッカソン運営者や参加者へのインタビューを通して，「チームメイトが互いを知り，肯定し合いながらゴールを迎えること」，「企画にあった内容を全てプロダクトに落とし込めること」，「プレゼンテーションで実物を見せられること」などの解答が得られた．ただし，「各参加者の動機にも左右される部分が大きく，明確な指標を立てることは難しい」との意見も述べられている．本研究で焦点を当てている成果物重視型のハッカソンにおいては，企画内容を忠実に実現した成果物の完成と，プレゼンテーションにて成果物の内容を伝えることは重要な目標であると考えられる．このため，ハッカソンチームの成功指標には以下を設定した．

- 企画内容に忠実な成果物を実現できたか
- プレゼンテーションにて成果物の内容を十分に伝えられたか

第 3 章

関連研究

共同開発の支援として，これまでに様々なプロセスやシステムが提案されており，Agile や SCRUM を筆頭として，小規模の組織での運用に焦点を当てたプロセスモデルは近年増加している [16][17][18]．また，プロジェクトメンバ同士のコミュニケーションログの可視化 [19] などをはじめ，組織内における情報共有の支援は古くから研究されている分野である．しかしながら，従来の支援システムのほとんどは，長期的なプロジェクトや，企業など確立された組織での活用を前提としたものが多く，企画・開発・プレゼンテーションを内包した濃厚かつ高速な作業や，即席のチーム結成が想定されるハッカソンへの導入に焦点を当てた支援手法の研究事例は少ない．また，ハッカソンおよび同様の形態を持つイベントに関する研究報告は近年少しずつ増加してきている [5][2][20][21] が，ハッカソンに関する調査を実施した既存の研究事例においては，各ハッカソンチームの内部で起きていた事象については明らかにされていない点が多い．本研究においてハッカソンチームの失敗要因を調査する上では，ハッカソンチーム内部での活動に注目する必要があった．

3.1　共同開発プロセスに関する研究

ソフトウェア開発における既存の共同開発プロセスはドキュメント駆動開発や Agile をベースとし，これまでに多くの派生プロセスが提案されている．

■ドキュメント駆動開発　要件定義書，基本設計書，詳細設計書といったドキュメントの作成を経て開発を進める．開発においては作成されたドキュメントをプロジェクト進行の主軸に置き，これを厳守する．これをベースとした派生アプローチは以下が代表的である．

- ウォーターフォール

システム全体を一括して管理し，分析・設計・実装・テスト・運用を順に実行していく．各工程が完了する際には，逆戻りが発生しないように綿密なチェックを行う．

- RUP（Rational Unified Process）
  ユースケース別に短期間のウォーターフォールを繰り返しながら，製品の機能を段階的に高めていくプロセスである．
- RAD（Rapid Application Development）
  プロトタイプを何度も制作，評価し，プロトタイプを次第に完成品に近づけてゆく手法である．
- スパイラル
  システムの一部分ずつをプロトタイピングし，顧客からのフィードバックやインターフェースの検討などを経て，さらに設計・実装を繰り返していく手法である．

■Agile 「あらかじめ設定した全ての工程が正しい」という前提で進行するドキュメント駆動開発に対し，Agile は「プロジェクトは常に変化する」という前提をとり，システムの機能を細かく区切った上で分析，設計，実装，テストなどのサイクルを反復的に繰り返す．たくさんの文書を作成することよりも，プロジェクト関係者間で必要な時に即座に直接顔を合わせて意思疎通を行うことを強調する．これをベースとした派生アプローチは以下が代表的である．

- XP（eXtreme Programming）
  初期段階の設計よりもコーディングとテストとフィードバックを重視し，開発者間の円滑なコミュニケーション，必要最小限の設計しか行わないシンプルさ，頻繁なテストによるフィードバック，大胆な設計変更に立ち向かう勇気を基点とした開発理論である．
- SCRUM
  技術的要素が取り除かれ，多くのチーム作業に対し汎用的に適用できる要素を残したチーム開発の枠組みである．「伝統的なリーダーが行ってきたことを，できる限りチーム全体が行うこと」をスタイルとし，要求される機能や技術的改善要素に優先順位を記述したプロダクト・バックログと，スプリント期間（1〜4 週間）分のチームのタスクを記述したスプリント・バックログによって開発を管理する．

3.1.1 ペーパープロトタイピング

アプリケーションの設計段階において古くから活用される手法として「ペーパープロトタイピング」が存在する．ペーパープロトタイピングは紙ベースでアプリケーションの動きをシミュレートする手法である．アプリケーションを構成する各画面を手書きのスケッチなどで設計し，各画面を部品ごとに切り分け，その後ペーパープロトタイプの作成者はアプリケーション役，その他のメンバはユーザ役に分かれる．ユーザ役はアプリケーションの画面に見立てられた紙の部品に触り，アプリケーション役はユーザ役のジェスチャに応じて紙の部品を操作する．紙とペンを用いて作成するペーパープロトタイプは高度なテクニックに依存しない上，即興性に優れているものの，画面の関連性を十分に説明できるのはプロトタイプの作成者本人に限られ，プレゼンテーションの再現性が保障されない問題がある．また，手書きのスケッチのような仮のプロトタイプがもたらすユーザエクスペリエンスは，現実のアプリケーションが提供するユーザエクスペリエンスとは異なるものとなる懸念も指摘されている [22]．HackathonMediator の「成果物イメージの具体化と共有を支援するモックアップ作成機能」においては，モックアップの画面構成に関する情報を実際のアーキテクチャ環境で動作可能なプログラム群に変換する機能を提供しており，実質的なアプリケーションとしてテストすることが可能である．これにより，使用者は作成したモックアップから確実なユーザエクスペリエンスを得ることが出来る．

3.1.2 Mockup Driven Development

Rivero ら [23] は，開発プロセスにおいて開発者とクライアントの統合性の強化を目的とし，開発者とクライアントの間の共通言語としてモックアップ作成ツールを用い，段階的なモデリングを可能にする Agile ベースのモデル駆動型開発アプローチ Mockup Driven Development を提案している．このアプローチでは，第一段階にて抽象度の高いプレゼンテーションモデル（モックアップ) を作成したのち，同著者らが開発したモックアップファイル解析ツール Mockup Processing Engine（MPE）を用いて Stractural UI モデルへモックアップを変換する．Stractural UI モデルは，画面遷移やコンテンツ仕様を豊富に記述したクラス図のような形態を持つモデルであり，モックアップ作成時において，アプリケーション上の動作やデータのヒントを示す「タグ」をモックアップ上に配置しておくことで，Stractural UI モデルに対し動作やデータ構造の記述を埋め込むことが可能である．タグは特定のウィジェット上のテキストパラメータとして記述しておき（図 3.1），MPE がモックアップファイルを Stractural UI モデルに変換する際に解析され，適切な形に置き換えられる（図 3.2）．

使用される主なタグは以下の 3 つである．

- Node（<nodeId>）
  ページ間のリンクを参照するために，各ページにユニークな ID を割り当てる．
- Link（<nodeId>）
  リンクやボタンなどのウィジェット上に適用可能であり，ウィジェットのデフォルトアクションとして Node(<nodeId>）がタグ付けされたページへのナビゲーションを定義する．
- Data（<elementType>）
  アプリケーションのコンテンツ仕様を記述する上で，基礎となるウィジェットや<elementType>間の関連を指定する．

同著者らは，モックアップから変換された Stractural UI モデルから，アプリケーション上のそれぞれのページ・リンクおよびクラスを推測しつつ開発者がコーディングでき，タグの組み合わせやメタ的な解釈次第で，クラス間のコンテンツ管理方法などを推測することも可能であるとしている．
このアプローチは，アプリケーションのデザインに関する議論のためにモックアップツールを使用し，その後モックアップを Stractural UI モデルに変換することにより，アプリケーションの実装作業への移行を迅速化する働きがあると考えられる．これは本研究が検討する「デザイナが作成したプロトタイプをプログラマが再利用できる形に最適化する仕組み」の一つと捉えられるものの，Stractural UI モデルは実装要件の明確化に留まっており，後述の TAP[22] と同様，開発段階への移行後にプログラマは一から実際のアプリケーションをコーディングする必要がある．

3.2　共同開発支援システムに関する研究

3.2.1　プロジェクトメンバ間のコミュニケーションログの可視化

Kwan ら [19] は，ソフトウェア開発プロジェクトにおいて頻繁に更新される要件はプロジェクト関係者間において常に共通の認識でなければならないとして，これを解決するためにプロジェクトメンバ間のコミュニケーションログ可視化システムを提案している．要件変更の際にプロジェクトメンバ間でやり取りされる情報の可視化問題などについて取り組み，プロジェクト進行において支援が必要とされる点を挙げている．

1. 個々の役割を可視化すること

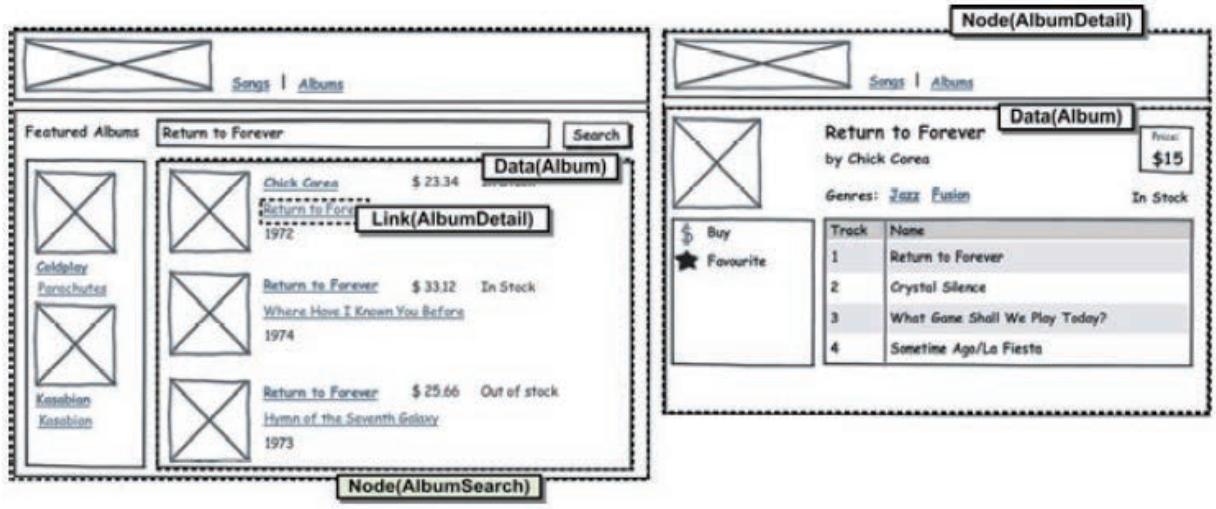

図 3.1　タグが付けられたモックアップ

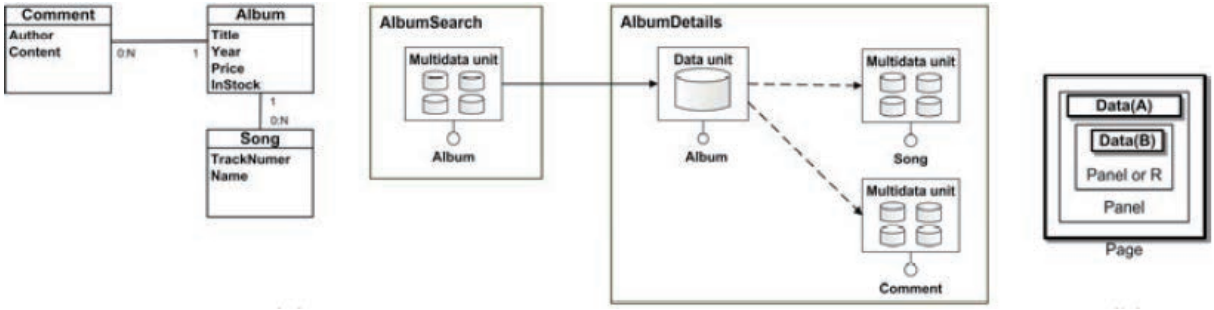

図 3.2　タグが反映された Stractural UI モデル

要件へのアクセス権を持つデザイナ，プログラマ，テスタ，マネージャ，バイスタンダーなどの役職を可視化する．提案システムにおいて，各個人をグラフ上に配置されたノードとして表し，ノードに対して異なる形と色を使用することで判別できる．

2. コンタクトの動きを可視化すること
   提案システムにおいて，個々のノードに対し通信フローの存在と方向を示す矢印がグラフ上で表される．矢印の線幅はコミュニケーション活動の数を表し，線の色は使用された通信媒体のタイプ（例：電子メール，チャット）を示す．
3. やりとりされる情報の詳細について容易にアクセスが可能であること
   提案システムにおいて，グラフ上のノードの配置は，最新の通信ログに依存して決定される．より最近行われたやり取りに関わるノードが中心近くに示される．ユーザは，グラフ上のノードや線をクリックすることで，個々のやり取りに関する詳細情報

にアクセスできる．

プロジェクトメンバのコミュニケーションはグラフによって可視化され，要件変化に影響を及ぼしたやり取りの経緯を把握することができるほか，同じ要件に取り組んでいるチームメンバに要件の認識を促す通知を送ることができる．
このシステムにおいては電子メールなどの通信媒体の積極的な組織内での活用が前提とされており，ハッカソンのような激務が及ぼす「チームのコミュニケーションが滞りがちな状態」においての適用は難しいと考えられる．ハッカソンにおける支援としては，まず第一にチーム内でのコミュニケーションを引き出す仕組みが必要だと考える．

3.2.2 Touch Application Prototype

Jørgensen ら [22] は，iPhone アプリを設計する業務を担うデザイナに向け，インタラクティブで現実的な iPhone アプリのプロトタイプを作成可能にし，実際のデバイス上でそれらをテストするためのツールとして Touch Application Prototype(TAP) を提案している．TAP を使用するデザイナは，アプリケーションのコンセプトができた時点で，ペーパープロトタイプや型紙（ステンシル）ベースでのワイヤフレームの構築などに一から取り掛かったのち，最終的に Adobe 社の Fireworks を使用してインタフェースをデジタル化する．Fireworks 自体は Web アプリケーションのグラフィック作成に特化したソフトであるが，TAP は Fireworks で作成したグラフィックインタフェースに対し，タップによって新しいページにリンクするアクティブ領域（ホットスポット）およびページ遷移のアニメーションを割り当てることができる．この機能によりデザイナはコーディングを必要とせずに，クリック操作が可能なアプリケーションプロトタイプを提示できる．Jørgensen らは，プロトタイピングツールに望まれる要素として以下のようなものを定義している．

1. 迅速かつ簡単に作れること
2. 実際のシチュエーション (利用者の生活の中でのワンシーン) を想定してテストできること
3. 実際のハードウェア上で見せることが可能であること
4. デザインの説明にファシリテータの存在を必要としないこと
5. 現実に対する高い忠実度と状態遷移を可能にすること

4 について Jørgensen らは，「ペーパープロトタイピングなどの抽象的な設計法は，通常はインタフェースの状態変化を手動で実行するファシリテータが必要であるが，デザイナの説明に他の人間を干渉させることは目障りであり，結果に否定的な影響を与えることがあるた

め，少なくとも一定の干渉を受けずにプロトタイピングできることが望まれる」と述べている．また，5 については，「ユーザが iPhone の標準インタフェースに慣れているのならば，スケッチのような抽象度の高いプロトタイプに違和感を覚える．また，iPhone のようなデバイスにおいては画面の描画領域が限られており，そのタッチ領域は指先で操作するのに十分な大きさでなければならず，したがってプロトタイピングツールは，現実的なインタフェースを使用可能にすべきである」と述べている．

TAP で作成したプロトタイプはデザイナのプレゼンテーションに特化しており，まるで本物のアプリケーションが動いているかのように錯覚させるという意味で，Smoke-and-Mirrors[24] のカテゴリに分類されている．このプロトタイプを実際のアプリに直接適用することはできず，実装に移行する上では一からのコーディングが求められる．ハッカソンにおいては実装に費やすことのできる時間の確保が重視される傾向にあり，デザインに関する議論は最小限に留められることが好ましい．そのため，ハッカソンの支援システムにおいては，デザイナが作成したプロトタイプをプログラマが再利用できる形に最適化する仕組みが必要であると考える．

3.3　ハッカソンおよび同様の形態を持つ取り組みに対する研究

3.3.1　F-Secure 社開催のハッカソンにおける研究報告

Raatikainen ら [20] は，F-Secure 社内においてハッカソンを調査・導入し，その結果から学んだハッカソンの運営・実用性・社会的意義などに関する教訓を述べている．同著者らは，ハッカソンの開催意義に関して「開発の高速性が不可欠になってきているソフトウェア工学における有望な新しいアプローチである」としており，「様々な目的にモチベーションを提供するため，社内で有効に活用できると思える．コラボレーション・インスピレーション・作業意欲が参加者の間で強調され，開発したアプリケーションは会社の直接的な利益となった」と述べているほか，ハッカソンの参加者に対し自社の API を提供したことにより，「API のボトルネックを明らかにし，今後の自社製品開発の方向性を提供する」効果があったと述べている．ハッカソンの開発スタイルは通常自社で過ごすスタイルと大きく異なり，シビアな時間制限や異なる事業に属する者同士のコラボレーションなどが非日常的な体験をもたらし，参加者の作業意欲を高める傾向にあることが示唆された．また，同著者らは，ハッカソンについて調査する上で「ハッカソンは人気であり，いくつかの技術革新がハッカソンやハッカソンに由来する経験から生じたとされるにも関わらず，その詳細についてはほとんど報告・議論されていない」と述べ，そのようなハッカソンを理解する上での研究アプローチは，定性的および探索的なものとなったとしている．

Raatikainen らの調査はハッカソンを科学的に評価する上で希少かつ先進的な報告事例と見なせるが，報告されている教訓はハッカソンの運営者からの視点に留まっている．

3.3.2　ゲームジャムにおける研究報告

ハッカソンのスタイルを内包しつつゲーム制作に特化した派生イベントとして，ゲームジャムが存在する．Musil ら [21] は，ゲーム開発を発展させる有望な選択肢としてゲームジャムに着目し，時間制限と競争的環境のあるゲームジャムにおいてもたらされるポジティブな影響を調査した．同著者は，ゲームジャムの規則について以下の 7 つを列挙している．

1. 小規模のプロトタイプ（実験的なゲーム）としてゲーム産業に新しいアイデアを吹き込むこと
2. 参加者全員が共有しなければならない一つのテーマが与えられる
3. 誰もが参加でき，誰もが新しいゲームを作った当事者となることができる
4. 24〜48 時間程度の時間的制約がある
5. チーム形成が促され，チームサイズは 2〜5 人程度でなければならない
6. ソフト・ハードを問わず最も熟達しているプラットホームとツールで展望を得ることができる
7. 終了後，作品の中から最も素晴らしい製品を選ぶ公開プレゼンテーションがある

これらの規則から，ゲームジャムとハッカソンはほぼ同じスタイルを保っていることが分かる．ただし，ゲームジャムにおいてこれらの規則は徹底されやすい傾向にあるものの，ハッカソンにおいては主催団体の意向などによりやや変動する規則（3，6 など）が存在する．ゲームジャムにおいては開発対象となるカテゴリをゲームに限定している点から，主催団体間で共通の開催スタイルを維持しやすくなっていることが推測されるが，ハッカソンにおいては主催団体間で共通の開発対象カテゴリを持たないため，開催スタイルが主催団体によってローカライズ化されがちであることが考えられる．また同著者らは，ゲームジャムで成功するチームの条件について，以下のように定義している．

1. サポートに優れたメンバが存在する
2. 急速なプロトタイパーが存在する
3. 企画内容の当事者が存在する
4. 異なる役職間を仲介することができるメンバが存在する

開発スタイルの類似性から，ゲームジャムの成功要因はハッカソンの成功要因に近いものだと推測できる．しかしながら，ゲームジャムと同様に，ハッカソンにおけるチームはアドホックな形で結成されることが多く，これらの条件を十分に満たす人員をチームに集めることは困難であると考えられる．そのため本研究においては，「急速なプロトタイパー」および「異なる役職間の仲介者」，「他メンバのサポートの促進」などの役割をハッカソン支援システムが担うことにより，チームを成功へと導く要因を補強する．

3.3.3　ペアプログラミングにおける研究報告

ペアプログラミングは，Agile アプローチにおけるエクストリームプログラミング (XP) に含まれ，2 人のプログラマが 1 つのキーボードを共有して共同作業を行う．プログラマは状況に応じて指示等を出すナビゲータと，コーディングを行うドライバーに分かれる．Zarb ら [25] は，ペアプログラミングにおけるベストなコミュニケーションガイドラインを提案している．

ペアプログラミングにおいては共同作業を行う中で即時のコミュニケーションを頻繁に取ることが推奨されており，これはハッカソンにおいても共通する要素である．ただし，ほとんどのハッカソンにおいてはチームメンバが 3 人を超え，メンバの職種もプログラマに限定されない場合が多い．ハッカソンにおけるコミュニケーションの課題は，ペアプログラミングにおいてこれまで検討されてきた課題がより複雑に変化したものと考えられる．

第 4 章

予備実験

HackathonMediator を検討・開発するにあたって，ハッカソンにおけるチームの失敗要因およびその解決手法を明らかにするための予備実験を行った．この予備実験は 4 度にわたって繰り返し実施しており，ハッカソンの参加メンバは実験毎に異なる．各ハッカソンにてチームの行動を観察し，ハッカソンチームの失敗要因として考えられる問題を発見していく．また，2 回目以降のハッカソンでは，過去に確認された失敗要因に基づいて考案した解決手法を導入しており，その手法の効果についても述べる．各実験において著者は「ハッカソンの運営者」として振る舞い，ハッカソンの最中に気になった行動などについてはチームにインタビューを行っている．なお，運営者に対し，チームが企画に関する感想や意見を求めたり，技術に関する相談をすることは許可されている．ただし，チームの失敗要因を明らかにする上で，チームマネジメントに関して運営側から積極的に意見することは禁じていた．

4.1　2014 年 4 月 19 日 8 時間ハッカソン

4.1.1　参加者

1. プログラマ A
2. プログラマ B
3. グラフィック・デザイナ（チームリーダ）
4. プランナ（プレゼンテータ）

参加チームは 1 チームであり，プログラマ 2 名，デザイナ 1 名，プランナ 1 名の計 4 名で構成される．なお，参加者の過去のハッカソン経験について，プログラマ A とデザイナが「一

度経験したことがある」と答えている．また，デザイナはチームリーダ，プランナはプレゼンテータの役割をそれぞれ兼任していた．このチームのマッチ度については，各メンバから「コミュニケーションが取りづらい雰囲気ではなかったが，全体的に静かすぎるチームだった」，「技術に秀でた人がいなかった」などの意見が述べられている．

4.1.2　手順

初回のハッカソンにおいては，実際のハッカソンにてチームがどのような行動を取っているのか，それらの行動が結果にどのような影響をもたらすかを観察した．ハッカソンの最中において主催者側からの指示は作業の開始・終了時刻とプレゼンテーションのタイミングに留まっており，企画段階および開発段階の時間配分について主催者側は指示せず，チームで思いのままに作業を進めてもらう形で進行した．

4.1.3　結果

企画段階において各メンバは積極的に発言する姿勢を見せ，アプリケーションのコンセプトの決定には全員が同意を示した．開発中盤ごろより，各チームメイトが担当する作業の内容と進捗状況が共有されなくなり，チーム内で誰が何をやっているのか把握できない状態に陥っていたことが確認された．確認された問題として以下のようなものがある．

1. 情報共有の頻度低下
   「開発が進むにつれて，情報共有の機会が大きく減った」との意見が多くあった．このことに関して，チームへのインタビューでは「作業に追い込まれていて，周囲とコミュニケーションをとる余裕がなかった」，「情報共有が必要だとは思っていたが，作業が進むにつれ周囲も忙しくなり，どこまでの情報を共有して良いか分からなくなった」などの理由が述べられている．
2. 共有された情報の把握不足
   このチームは，作成したファイルや参考にする Web ページなどを共有するためのツールとして，Facebook メッセンジャを利用していた．作成した素材や開発に役立つ情報などをツール上で共有する場面があったが，共有された内容についてすぐに確認しようとしたメンバはいなかった．このことについて，メンバはインタビューの中で「作業に夢中で気づかなかった」，「何かが共有されていたことには気づいていたが，先に作業を済ませてから確認しようと思った」と解答している．
3. 進捗状況の把握不足

プレゼンテーションの 10 分前にプランナが資料作りを開始するも，プランナは他メンバの進捗状況を把握できておらず，プレゼンテーションでは企画内容について述べるに留まり，開発したアプリケーションの説明には至らなかった．なお，プレゼンテーション時点において，他メンバはプランナが作成した資料の内容を認知していなかった．

4. 作業内容の重複
   開発段階においてデザイナによりおおまかな作業分担が行われたが，担当メンバ・担当でないメンバが同一の作業に着手していたことが複数回確認された．そのことについて指摘された際，担当でないメンバは「分担された作業範囲をよく理解していなかった」と述べている．
5. 進捗状況の把握不足
   機能の削減などに伴い，デザイナは該当機能の実装を担当するメンバに対し作業内容の変更を伝えようとしたが，その際「誰がどの部分を担当しているのか」を把握できていなかった．デザイナに限らず，多数のメンバがインタビューにおいて「周りのやっていることが見えていない状態だった」と述べている．
6. 具体性を欠いた指示
   プランナはグラフィック・デザイナから「参考になるようなアプリケーションのデザインを探して欲しい」と依頼を受けたが，その後 2 時間近く具体的なデザインを提示できない状態が続いた．プランナは後のインタビューにて「ネットでデザイン例を調べるなどしていたが，どのようなデザインが求められているかなどの具体的なことが分からず立ち往生してしまった」と述べている．

4.1.4 考察

今回確認された進捗状況の把握不足（5），作業内容の重複（4），進捗状況の把握不足（5）などの問題は，情報共有の頻度低下（1）や共有された情報の把握不足（2）が原因となって生じていると考えられる．また，具体性を欠いた指示（6）の問題については，プランナに対するインタビューを通して，チームメンバがアプリケーションのイメージについて深く理解していなかった可能性が推測された．このチームはアプリケーションのアイデアに関する議論を終えた直後から開発に取り掛かっており，アプリケーションの完成イメージや，それに基づくメンバの担当区分について事前に十分な議論を行わなかった点が見られた．

10 tips for hackathon success[26] の中において，ハッカソンの成功の秘訣として「自分たちがデモンストレーションで最も示したいのは何か」を各チームメンバが理解するこ

とが重要であるという指摘がなされている．また，F-Secure 社でのハッカソンを調査した Raatikainen ら [20] は，「いくつかのチームでタスクが重複するなど無駄な努力が発生した」と述べ，「参加者は，今誰が何をしているかを全員が把握・理解し，全てのタスクに意味を見出した上で，お互いの責任分担に明確に同意できなければならない」としている．これらのことを踏まえ，次回のハッカソンにおいては，各メンバの担当区分とアプリケーションの完成イメージについて早期に議論する機会を設けるべきだと考えた．

4.2　2014 年 5 月 31 日 8 時間ハッカソン

4.2.1　参加者

1. プログラマ A
2. プログラマ B（チームリーダ，プレゼンテータ）
3. プログラマ C
4. グラフィック・デザイナ（プランナ）

参加チームは 1 チームであり，プログラマ 3 名，デザイナ 1 名の計 4 名で構成される．なお，参加者の過去のハッカソン経験について，プログラマ B のみ「一度経験したことがある」と答えている．また，プログラマ B はチームリーダとプレゼンテータ，デザイナはプランナの役割をそれぞれ兼任していた．このチームのマッチ度については，各メンバから「意見を出せる人があまりいなかった」，「仲は悪くなかったものの，特に良いとも言えず，意思疎通があまりとられないチームだった」などの意見が述べられている．

4.2.2　手順

今回のハッカソンでは，各メンバの担当区分とアプリケーションの完成イメージについて早期に議論する機会を確保するために，チームに対し企画終了段階でプレゼンテーションの資料作りを行うように指示した．早期の段階でプレゼンテーション資料を作成することにより，各チームメンバに対し共通の目標設定をもたらす作用を期待した．作成する資料には，各々が担当する作業の分担や，アプリケーションの完成イメージなどを詳細に記述するように促している（図 4.1）．

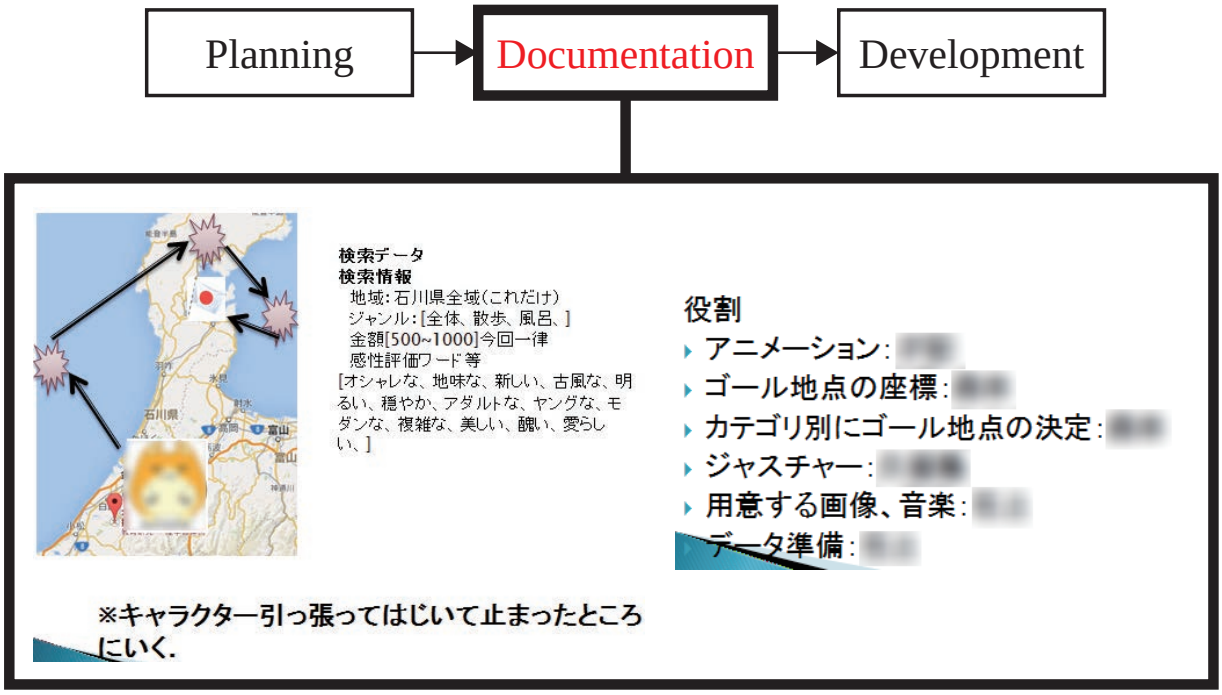

図 4.1　プレゼンテーション資料の一部

4.2.3　結果

「プレゼンテーション資料を事前に作ったことで開発に影響はあったか」という質問に対し，全てのチームメンバが「効果はあった」と述べ，「作ろうとするもののイメージはしやすかった」，「資料通りにアプリケーションが完成していった」，「開発の見通しが良くなった」としつつも，プログラマ A からは「必ずしも全てが資料通りにはならなかった．状況に合わせて変化していった仕様もいくつかある」，「資料の内容について理解できない部分もあった．出来ていくアプリケーションを見ているうちに，ようやく内容を理解した」という意見があった．今回のハッカソンの最中に新たに確認された問題として，以下のようなものがある．

1. 成果物に抱くイメージの差異
   資料の内容について「出来ていくアプリケーションを見ているうちに，ようやく内容を理解した」という意見があったように，プレゼンテーション資料内に表記された一部の機能に対し，チームメンバの理解が曖昧なままだったことが確認された．
2. 後発的な作業の増加

企画段階において事前に把握されていなかった技術的な課題や作業内容が開発段階への移行後に相次いで判明した．

例：用意するデータの扱い方について議論していなかった，オブジェクトの衝突判定のアルゴリズムを検討していなかった，など

3. 実装後の修正要求

   プログラマ B が実装した機能をプランナが確認した際，「見栄えが良くない」とし，デザインや視覚エフェクトなどに関して新たな要求を提示したものの，ハッカソンの時間内にその要求が反映されることはなかった．

4. 他メンバへの関心の希薄化

   プレゼンテーション資料はプランナがメインとなって作成しており，最初の時点では他メンバが作成中の資料に対し意見するなどしていたが，次第に時間を気にし始めたメンバが開発環境の整備などを並行して行うようになった，その後プランナが完成した資料を提示するまでの間，他メンバが資料に注目することはなくなった．

5. ソロプレイヤー化

   あるメンバは自分の担当する作業の中で問題を解決できずにいたが，他メンバに協力を求めることをしなかった．

   例）プログラマ C は数時間に渡って他のメンバとコミュニケーションを取ることをしなかった．後にプログラマ C の作業が進んでいないことが発覚し，別のプログラマが作業を手伝うこととなった．このことについて，プログラマ C は「一人になって意固地になってしまっていた」と述べている．

6. 技術的な問題発覚

   「苦労した点」に関するインタビューの中で，プランナは「できるだろうと思ってプログラマに指示を出したが，結構な頻度で反発が返ってきた」とし，「こちら側の理解とプログラマ側の理解のすり合わせが全然うまくいってなかった」と述べている．

7. 開発への焦り

   他メンバへの関心の希薄化（4）でプランナの作業が注目されなくなった理由として，プログラマらは「開発に残された時間を気にしていて，できるだけ早く作業を進めておきたかった」という旨を述べている．

8. リソースの未使用

   プランナは多くのテキストデータや画像素材などを用意したが，最終的なアプリケーションに反映されたものはごくわずかだった．テキストデータに関してはそれを扱うための機能そのものが設計されておらず，プログラマは「機能を実装する余裕はなかった．そもそも何に使うデータなのかをよく理解していなかった」という旨を述べ

ており，またプランナはインタビューにて「時間をかけて用意した素材が使われないのは悔しかった」と述べている．

4.2.4 考察

今回の実験では，プレゼンテーション資料の早期作成を促すことにより，各メンバに対しプロダクトと作業内容に共通認識をもたらす作用を期待した．しかしながら，作業内容の重複や実装内容の誤解などが発生していることから，大きな効果は得られていないと考えられる．当初の指示において，プレゼンテーション資料には「作業の分担，アプリケーションの完成イメージなどを詳細に記述するように」と促した．しかし，「理解できない部分もあった」ことがインタビューにて指摘されているように，実際に記述されたプレゼンテーション資料の内容は，プレゼンテータの説明を多分に必要とする抽象度の高い内容であった．この理由として，開発への焦り（7）などの問題が大きく関係したと考えられ，その結果，成果物に抱くイメージの差異（1）がチームメンバ間で発生したのではないかと推測される．チームメンバはハッカソンの時間的制約に関して強く意識しており，チーム一体での議論が求められる状況においても個人の作業を優先しようとした結果，プレゼンテーションの資料作りに十分な貢献ができなかった．彼らはプランナがほぼ一人で完成させた資料を後になって確認することになり，曖昧な理解を含んだまま作業に取り掛かってしまったと考えられる．

■「最小限の議論時間」の検討　実験を通して，プレゼンテーション資料の早期作成は「後の開発過程における見通し」を明らかにする上で有効性が期待されたものの，開発を控えた状態の中でチームを長く議論の場に拘束することは困難であると思われた．そのため，次回のハッカソンでは「最小限の議論時間」を考慮しつつ問題の解決手法を検討する．ハッカソンチームでの作業分担においては「タスクに意味を見出す」ことが重要であると Raatikainen ら [20] は述べており，そのためには「アプリケーションの完成イメージの明確化」が不可欠であると考えられる．今回のハッカソンにおいては，アプリケーションの仕様をはじめユーザエクスペリエンスなどに関連する詳細な情報をチーム内で十分に明確化できておらず，各メンバが分担された作業内容に意味や関心を見出せなかった可能性が高かった．これらのことから，次回のハッカソンでは，アプリケーションの仕様やユーザエクスペリエンスなどを明確化する手法として，ペーパープロトタイピングに着目した．

4.3　2014 年 7 月 19 日 10 時間ハッカソン

4.3.1　参加者

1. プログラマ A
2. プログラマ B
3. グラフィック・サウンド・デザイナ
4. プランナ（チームリーダ，プレゼンテータ）

参加チームは 1 チームであり，プログラマ 2 名，デザイナ 1 名，プランナ 1 名の計 4 名で構成される．なお，参加者の過去のハッカソン経験について，プログラマ A が「二度経験したことがある」，プログラマ B が「一度経験したことがある」と答えている．また，プランナはチームリーダとプレゼンテータを兼任していた．このチームのマッチ度については，各メンバから「プログラマの主張が強くなりがちだった」，「あまり協調性は良くなかったが，最初と最後の作業に関しては互いに協力できていた」などの意見が述べられている．

4.3.2　手順

今回のハッカソンでは，アプリケーションの詳細仕様やユーザエクスペリエンスなどをチーム内で明確化する上で，ペーパープロトタイピングを導入した（図 4.2）．企画段階が終了し，アプリケーションのコンセプトに関する議論を終えた時点で，アプリケーションに抱くイメージをペーパープロトタイプとして各メンバがそれぞれ具体化する．その後各自がチームの前でデモンストレーションを行うことで完成イメージの差異を吸収し，目標設定の共通化を図った．

4.3.3　結果

企画段階の終了後に実施したペーパープロトタイピングにて，各メンバが作成したペーパープロトタイプは画面構成や機能がそれぞれ異なっており，企画段階にて議論されていなかった機能の追加をはじめ，「アプリケーションの画面の数」，「画面の順序」などが新たに議論された．ペーパープロトタイピングを通して，各メンバからは「初期段階のイメージ共有としてはとても良いと思った」，「みんな考えてることが全然違うことに気づかされた」という感想が述べられている．一方で，「結局，そこから文章に起こすことをしなかったので，開発中に忘れてしまった部分もあった」，「資料をしっかり作って意思をあわせる工程が入る

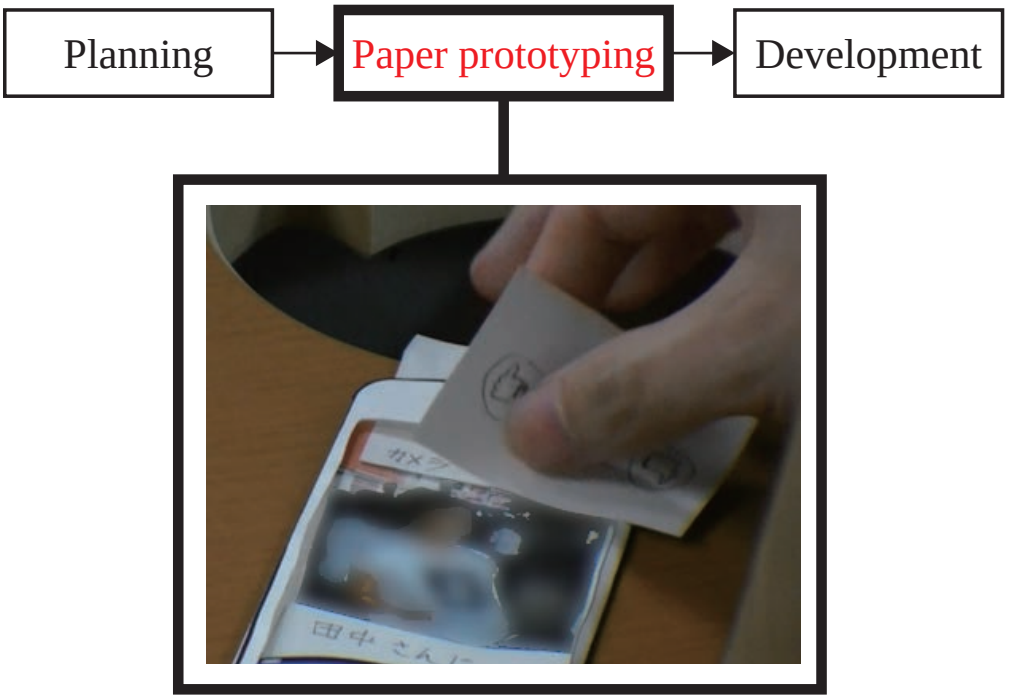

図 4.2　ペーパープロトタイピングの様子

ともっと良かった」などの意見もあった．

ペーパープロトタイピングにより，アプリケーションの完成イメージを明確化できた旨の感想が得られた．しかし，開発過程においては長時間に及ぶバグの修正作業などが発生し，対応していく中で大きなスケジュール変更が発生した．今回のハッカソンの最中に新たに確認された問題として，以下のようなものがある．

1. 想定外の作業の発生
   開発過程において，長時間に及ぶバグの修正作業の発生や，技術的に実装が困難な機能の存在が発覚し，対応に見舞われた結果大幅なスケジュール変更が発生した．
2. 待機状態化
   プランナとデザイナはハッカソンの終盤において「予定の作業が終わったが，次に控えている作業を把握できておらず，何をすればいいか判断できなくなった」と述べている．彼らの作業内容は「必要となる画像素材の確保」だったが，素材をプログラマに渡せないまま，プログラマの作業が終わるタイミングを待ち続けていた．
3. 優先順位の誤り
   開発後半において，チームは実装要件の再チェックのためにペーパープロトタイプを再確認した．その際，アプリケーションのコンセプトに直接関係のない機能に対し，

長時間実装に注力していたことが発覚した．

4.3.4　考察

ペーパープロトタイピングにより，各メンバがアプリケーションに抱く完成イメージが表出化され，チームの目標設定を共通化する作用が見受けられた．しかし，「開発中に忘れてしまった部分もあった」という意見などから，ペーパープロトタイピングによって表出化された完成イメージは，開発過程における想定外の作業の発生（1）などの要因によって有耶無耶になってしまっている可能性があると考えられた．優先順位の誤り（3）などの問題は，「実際のアプリケーション」と「完成イメージ」間の乖離が発生する中で誘発されたと思われる．したがって，次回のハッカソンにおいては，想定外の作業の発生（1）時などに，チーム内で迅速な認知と意思決定を促すための「タイムリーな情報共有手段」を検討した．

4.3.5　ペーパープロトタイピングの欠点

チームメンバが成果物イメージを共有する上では，ペーパープロトタイピングのような手法を用いることが有効であるとみられた．しかし，ペーパープロトタイピングには以下のような問題も見受けられている．

1. ペーパープロトタイパーの表現力の個人差
   個人別に作成したペーパープロトタイプにおいて，チームメンバからもっとも評価が良かったのは，印刷した画像を貼り合わせて作られたプロトタイプだった．プロトタイプを確認したメンバからは，手書きのものに比べて「アプリケーションのイメージがはっきりしている」という発言が得られている．また，インタビューでは「どの程度詳細に説明すればいいのか判断できず，非常に簡素な説明になってしまった」，「プロトタイプの作りこみが良いほど，メンバからの同意も得られやすくなると思った」などの感想が述べられた．
2. ペーパープロトタイプの再現性
   ペーパープロトタイプの再現性は保障されない．「優先順位の誤り」の発覚時にチーム内でペーパープロトタイプを再確認した際，チームはプロトタイパーの説明を再び必要とした．ペーパープロトタイプそのものはスケッチの集まりに過ぎず，「状態遷移のイメージ」はプロトタイパーの頭の中にのみ存在する．

以上の点から，成果物イメージを共有する上でより良い方法は，「再現性」を保証するためにデジタルインタフェース上でプロトタイピング手法を実現し，「表現力の個人差」を吸収するためにアプリケーションの基礎的な UI パーツをテンプレートとして提供し，一つのプロトタイプをチームが共同編集できることであると考えた．

4.4　2014 年 8 月 2-3 日 30 時間ハッカソン

4.4.1　参加者

- チーム A
  1. プログラマ A（チームリーダ）
  2. プログラマ B
  3. プログラマ C（プランナ，プレゼンテータ）
  4. プログラマ D
  5. グラフィック・デザイナ
- チーム B
  1. プログラマ A
  2. プログラマ B
  3. プログラマ C
  4. プランナ（チームリーダ，プレゼンテータ）
  5. グラフィック・デザイナ

参加チームは 2 チームであり，チーム A はプログラマ 4 名，デザイナ 1 名の計 5 名，チーム B はプログラマ 3 名，プランナ 1 名，デザイナ 1 名の計 5 名で構成される．なお，参加者の過去のハッカソン経験について，チーム A のプログラマ A，プログラマ B，プログラマ C が「複数回経験したことがある」，チーム B のプログラマ C が「一度経験したことがある」と答えている．また，チーム A において，プログラマ A はチームリーダ，プログラマ C はプランナとプレゼンテータをそれぞれ兼任し，チーム B において，プランナはチームリーダとプレゼンテータをそれぞれ兼任していた．チーム A のマッチ度については，各メンバから「技術を持っている人はいたが，連携がうまくいっていない」，「学内参加者と学外参加者の間で温度差があった」などの意見が述べられており，チーム B のマッチ度については，各メンバから「高専出身者が多いせいか，みんな仲が良かった」，「プログラマ A が技術的に突出しており，非常に助けられた」などの意見が述べられている．

4.4.2　手順

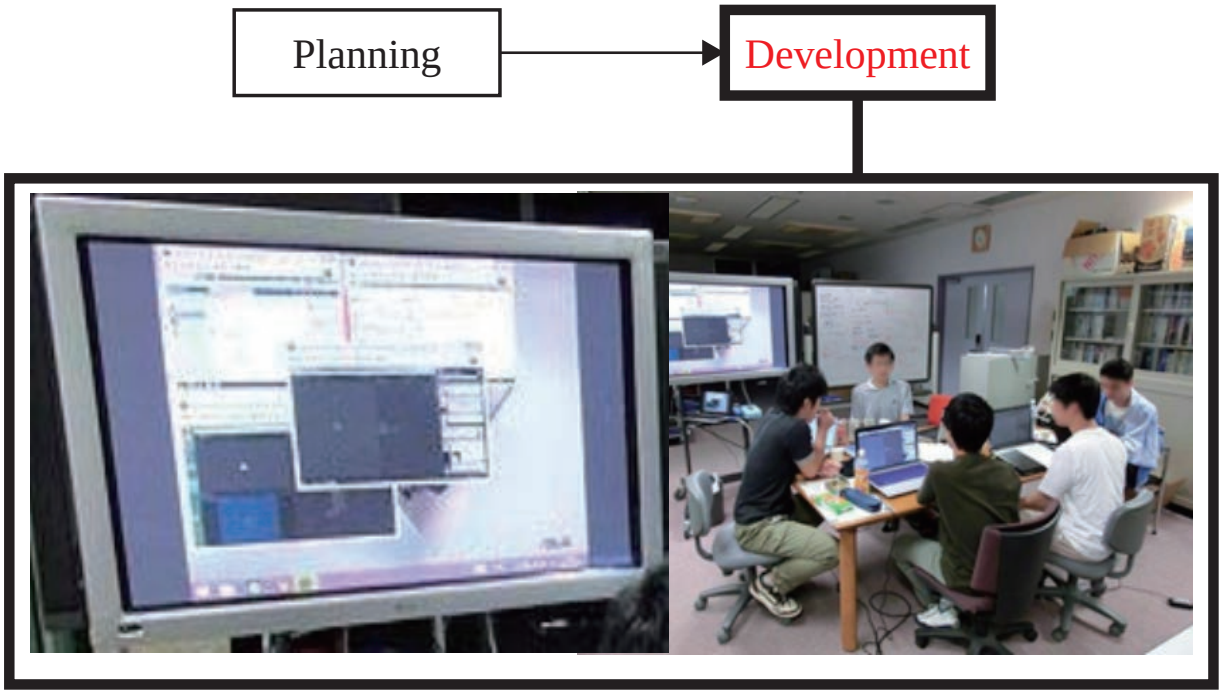

図 4.3　グリットレイアウト形式でミラーリングされた作業画面

今回のハッカソンでは，参加メンバ全員の作業画面を大型ディスプレイ上で常時ミラーリングする手法をとった（図 4.3）．大型ディスプレイ上では，各メンバの作業画面がグリットレイアウト形式で表示されており，全メンバの作業状況を一望できる．この手法を用いることにより，チーム内で「問題に囚われた状態」や「作業の完了」などを他メンバに対し迅速に認知を促す効用を期待した．

4.4.3　結果

開発初期段階において，大型ディスプレイにミラーリングされた作業画面を注目したメンバによる示唆の行動が多数見受けられている．しかし，開発が後半になるにつれて，作業画面に対するメンバの注目頻度は大きく減少した．参加者に対し実施したインタビューでは，「皆が何らかの形で作品に貢献している姿を確認できて面白い」，「行動できている人，できていない人を把握する上で役立った」などの好意的な意見が得られているものの，その一方で「気づかないことが多かった」，「メンバが何らかの作業を行っていることは分かるが，具

体的に何をやっていて，作業がどのような段階なのかは分からない」，「誰の作業画面なのかが直感的に分からない」などの指摘もあった．

4.4.4　考察

作業画面を常時ミラーリングする手法により，行動できている人やできていない人を把握する上で有効であるとの評価を得られた．しかし，「誰の作業画面なのかが直感的に分からない」という指摘があるように，共有された画面にはいくつかの情報を補足する必要があると思われる．また，「気づかないことが多かった」という意見があるように，画面を表示するだけでは「ソロプレイヤー化」，「他メンバへの関心の希薄化」などの問題を解決できていない．このようなことから，作業画面を常時共有するのではなく，「作業上の重要なタイミングで画面を自動的に共有する仕組み」と，その際に「各メンバに対し画面への注目を促す仕組み」が必要だと考えられた．

4.5　学外のハッカソンに関する現地取材

予備実験を行う過程で，「学外にて実際に開催されているハッカソン」に対する現地取材を行っている．以降では本研究で実施した「Mashup Hackathon 北陸 in 福井」での取材について述べる．

4.5.1　2014 年 8 月 30-31 日 Mashup Hackathon 北陸 in 福井

国内において，「Mashup Awards」と称する大規模な開発コンテストが毎年開催されている．このコンテストは「広く自由な発想に基づいた Web・スマートフォンアプリ等の開発作品」のエントリーを一般募集しており，アイデア，完成度，デザインなどを審査対象として，優れた作品に対し賞金または賞品が授与される仕組みとなっている．開催毎には IT 分野を初めとする多くの企業がスポンサーとなり，各社のサービスに関連した API などが無償で参加者に提供される．スポンサー企業においては独自に「企業賞」を用意する場合もあり，この賞は主に「自社の API を活用したプロダクト」の中から選出される．

Mashup Awards では，コンテストの一次予選の一環として「Mashup Hackathon」を開催している．Mashup Hackathon では，会場に集まった参加者でチームを結成し，「Mashup Awards への作品応募」を目標としたプロトタイプを 1-2 日の期間内（会場によっては 9 時間程度の場合もある）に開発する．今回，著者は 2014 年 8 月 30-31 日に福井県産業情報センタービルにて開催された「Mashup Hackathon 北陸 in 福井」を取材した．

■特徴

- 参加者

  20 名の参加があり，9 割近くが社会人の参加者であった．3 名程度の参加者は 2 日目から会場に現われており，各チームの助っ人として参加した．結成されたチームは当初 5 チームだったが，後にチームメンバが「自分のやりたいアイデア」を優先した結果，2 日目には 7 チームと単独作業者数名の構成になっていた．このことに関して「ハッカソンの最中にチームが分裂することは多いのか」という質問を運営関係者に行ったところ，「予定通りいかなかったり，別のアイデアにモチベーションが湧いた場合にはそういうこともある」としつつ，「この会場の参加者同士は旧知の間柄が多く，そのような選択も許される雰囲気だったのでは」という見解が述べられた．

  参加者の募集に関して，Mashup Hackathon では「エンジニア，デザイナ，ディレクタ」などの役職を持つ人々を優先的に募集しているが，「ハッカソンに興味のある方であれば誰でも」受け入れる姿勢をとっていた．

- 技術サポータの派遣

  Mashup Awards と同様，Mashup Hackathon においても複数の企業がスポンサーとなり，各社のサービスに関連した API を提供している．また，自社の API に関する説明や活用方法のサポートを担当する「技術サポータ」が一部の企業から派遣されており，ハッカソンの開催期間中において会場内に常時待機していた．

- ハイライト法によるチーム結成

  今回のハッカソンにおいては，チームの結成手法としてハイライト法が用いられていた．各参加者がアイデアシートを複数枚記入したのち，各アイデアシート内の好ましいアイデアに対し会場内の全員が投票（星付け）を行う．その後，星の数が最も多いアイデアシートから順に上位 6 点程度のアイデアが選出され，該当するアイデアシートを記入した参加者がプレゼンテーションを行う．この時，入選を逃したアイデアの中で記入者が「イチオシ」したいものがある場合も，特別にプレゼンテーションの機会が提供された．最後に，プレゼンテーションを行ったアイデアに対し「作りたい」と思うものに参加者が集まる．この手法により 5 つのチームが結成された．「ハイライト法を使ったチーム結成はハッカソンでよく行われるのか」という質問を運営関係者に行ったところ，「チーム結成と企画をスムーズに進行する上で用いられることはある」としつつ，「マジョリティに依存するため，ユニークなアイデアが弱くなってしまったり，人数が多い状況でないと導入が難しいため，万全な方法というわけではない」ということが述べられた．

■予備実験で確認された問題に関連する運営者・参加者の意見

- 「ソロプレイヤー化」について
  「作業に集中するなどで，他メンバに気を配れなくなることはあるか」という質問を数名に実施したところ，運営関係者は「かなりあると思う」と述べたほか，複数のエンジニアの参加者が「自分もそのような状態を経験をした」とし，「エンジニアは自分の作業に意地を張ってしまいがちで，他人に頼ることを避けようとする人も多いかもしれない」と述べた．
- 「情報共有の頻度低下」について
  「情報共有はできたか」という質問を数名の参加者に実施したところ，2 人で結成されたチームの参加者は「人数が少ないので，他のチームに比べてやりやすかったと思う」とし，「しかし，作業は非常に過酷だった」と述べている．6 人で結成されたチームの参加者は「人数が多い上に 2 日目から参加した人もいたので，全員が何をしていたかを正確に把握できた人はいなかったと思う」とし，「人数が多い分，プレゼンテーションに余力を割くことができたと思う」と述べている．
- 「想定外の作業の発生」について
  「想定外の作業の発生は多かったか」という質問に対し，複数の参加者は「ほとんどがそのような作業だったと思う」と答え，その理由として，「初めて使う API の仕様に慣れなかった」，「（認識系 API に関して）精度の問題などが後から発覚した」ことなどが影響し，「想定していた機能の実装を見直したり，企画を根本的に考え直したりもした」と述べている．
- 「技術的な問題発覚」について
  この問題について，運営関係者は「企画の際に “どのようにしてそれを実装するか” という内部ロジックに関する議論を行うこと」が重要であるとし，その解決手法として「アプリケーションの簡単なスケッチが作られる過程で，具体的な実装方法に関する議論を組み込む余地が設けられると良い」と述べた．

■参加者からの問題提起　ハッカソン会場の参加者に対し「ハッカソンに対して感じている問題はあるか」という質問を行ったところ，以下のような返答が得られている．

- 運営・サポータ・チームの間での遠慮の発生
  技術サポータは，「サポートしていたチームがとても集中モードで，“進捗どうですか” といちいち聞きに行くのも，ちょっかいをかけているようで悪い気がした」と述べて

おり，チームに介入することをためらう旨が示唆された．なお，そのチームのメンバからは「運営や技術サポータを頼りたかった面は沢山あったが，あまり迷惑をかけるのも申し訳ないと感じていた」という旨が述べられており，双方の立場において遠慮が発生していたことが確認された．また，ある参加者は「開発の途中で API の活用を中止することも検討したが，サポートしてくれた企業に申し訳ないと思い，言い出しにくかった．結果的にアプリケーションは中途半端なものになってしまった」と述べていた．

- 参加者の固定化
  参加者から，「例年のハッカソンを見ていると，仲の良い者同士が固まってきている．学生などの新しい参加者が増えてほしい」という旨が述べられている．また，「参加者が固定化されがちな理由についてどう思うか」という疑問に対し，「ハッカソンのイメージが “非常にハードな作業”，“エンジニアが参加するイベント” として定着してしまっていて，新しい人が寄り付かないのでは」という考えが述べられた．このことについて，運営関係者からも「地方でのハッカソンは未だ参加者が固定されがち」なことが述べられているが，「東京でのハッカソンも昔は同様だった」という前例から，今後は地方での参加者も増加していく可能性は大いにあるという見解が得られた．

4.6 ハッカソンで確認された問題のまとめ

予備実験において確認されたハッカソンチームの問題を「根本的な要因と見なされる問題」と「間接的に発生した問題」に区別した結果を，表 4.1 に示す．

多くの問題が発生した根本的な要因として，「開発への焦り」が考えられる．ハッカソンにおけるシビアな時間制限が，チームに対し「議論の時間を最小限に抑える」という観念を強くもたらしていることが考えられ，チームは「より細部に至るまでの議論」を省略してしまう．これに対しむやみに議論を長引かせようとすれば，メンバは開発に残された時間を気にし始め，チームの議論は「一体」ではなくなる．これにより，チーム内で「成果物イメージの差異」が発生してしまう．「成果物イメージの差異」は開発過程において「優先順位の誤り」や「実装後の修正要求」，「想定外の作業の発生」などの様々な問題を引き起こす．「想定外の作業の発生」などによって「後発的な作業の増加」が起こり，各チームメンバは作業に追い込まれ，「他メンバへの関心の希薄化」や「情報共有の頻度低下」，「共有された情報の把握不足」を引き起こしてしまう．結果として「進捗状況の把握不足」，「待機状態化」などが発生するほか，「優先順位の誤り」や「リソースの未使用」などの問題の発見が遅れることとなる．

表 4.1　根本的な要因と見なされる問題および間接的に発生した問題

根本的な要因と見なされる問題	間接的に発生した問題
開発への焦り	成果物イメージの差異
成果物イメージの差異	技術的な問題発覚
	実装後の修正要求
	具体性を欠いた指示
	優先順位の誤り
	リソースの未使用
	待機状態化
	想定外の作業の発生
想定外の作業の発生	後発的な作業の増加
	情報共有の頻度低下
後発的な作業の増加	優先順位の誤り
	リソースの未使用
	他メンバへの関心の希薄化
他メンバへの関心の希薄化	情報共有の頻度低下
	共有された情報の把握不足
共有された情報の把握不足	進捗共有の把握不足
進捗共有の把握不足	優先順位の誤り
	リソースの未使用
	ソロプレイヤー化
	待機状態化

4.7　解決すべき問題の検討

節 4.6 で述べた内容から，提案システムの支援要素として以下の二つを検討した．

1. **開発に遅延なく成果物の完成イメージを明確化・共通化する**
   「開発への焦り」と「成果物イメージの差異」の問題を解決することにより，「優先順位の誤り」や「技術的な問題発覚」,「実装後の修正要求」などの問題の発生を防止することができると推測された．この二つの問題を解決する上では，開発段階への移行

に遅延をもたらすことなく，成果物イメージを明確化・共通化する必要があると考えた．

2. チーム内での作業進捗の共有を確実なものにする

「他メンバへの関心の希薄化」，「情報共有の頻度低下」，「共有された情報の把握不足」などの問題を解決することにより，「進捗状況の把握不足」，「待機状態化」などの問題の発生を防止し，「優先順位の誤り」や「リソースの未使用」などの問題の早期発見を促すことができると考えた．これらの問題を解決する上では，チーム内での作業進捗の共有を確実なものにする必要があると考えた．

第 5 章

提案システム

提案システム「HackathonMediator」は，予備実験で検討した解決すべき問題に対し，「成果物イメージ共有を兼ねたプロトタイプ Ver.0 作成機能」および「作業進捗共有を兼ねた画面共有機能」を提供することによって解決を図る．本章では，このシステムを開発するにあたって検討した提案手法の概要について述べ，この二つの機能についてそれぞれ解説する．

5.1　提案手法の概要

提案手法は，主に企画終了時点と開発過程において用いられる．それぞれの段階における支援内容を以下に述べる．

- 企画終了時点での支援
  「開発に遅延なく成果物の完成イメージを明確化・共通化する」ために，この手法では，アプリケーションのスケッチの作成工程を「成果物イメージに関する議論」として完結させるのではなく，「開発段階での初期作業」として導入可能にする．企画段階の終了後，チームはアプリケーションの完成イメージについて議論し，一つのプレゼンテーションモデル（モックアップ）を共同作成し，実装に関する注釈（変数やメソッドの宣言，コメント文）をモックアップ内に記述する．プログラマやデザイナなどの異業種が同一の作業空間を有することにより，「成果物イメージの差異」を防止する．ここで作成したモックアップと注釈文は，後述する「成果物イメージ共有を兼ねたプロトタイプ Ver.0 作成機能」を経由してアプリケーションのソースコード（プロトタイプ Ver.0）に変換される．企画から開発段階への移行をシームレス化することにより，「開発への焦り」の抑止作用を期待する．チームはこの工程を適宜繰り返したのち，プログラマは，通信プロトコルを使った処理やアルゴリズムなどの「内部

機能」を実装し，デザイナは画像などのリソース作成，プランナはプレゼンテーション資料などの作成を開始する．

- 開発過程での支援
  「チーム内での作業進捗の共有を確実なものにする」ために，プログラマがアプリケーションのデバッグを開始した際や，デザイナやプランナが新規ファイルを作成・編集した際，後述する「作業進捗共有を兼ねた画面共有機能」がそれらのアクションを検知し，該当メンバの作業画面を大型ディスプレイにミラーリングする．これにより「情報共有の頻度低下」の問題を防止する．またこの時，「他メンバへの関心の希薄化」や「共有された情報の把握不足」を防止するために，「通知メッセージのレベル設定」機能（節 5.3.5）により，他メンバに対し作業内容の重要度に応じた通知メッセージを発行し，ミラーリングされた画面の確認を促す．

5.2 成果物イメージ共有を兼ねたプロトタイプ Ver.0 作成機能

本機能では，ハッカソンの企画段階の終了時に，各チームメンバがそれぞれ抱く成果物イメージの具体化・共通化のプロセスを支援する．この機能は Linux 系サーバ上の PHP スクリプト群によって実装されており，公開 URL にアクセスすることでマシン環境に依存なく利用出来る．本機能を活用した場合，利用者の活用手順は以下のようになる．

1. デザイナが画面構成およびレイアウトを定義したモックアップを作成する
2. プログラマが内部ロジック等の実装に関する注釈文をモックアップ内に記述する
3.「コンバート」ボタン経由でアプリケーションのソースファイルを出力する
4. 実行環境でソースファイルを実行し，デザインに関して反復的に評価・修正する
5. 注釈文の内容に基づき，内部ロジックに関する実装を施す

企画終了時において，まず最初にデザイナなどのメンバがアプリケーション画面のモックアップを作成する（1）．プログラマはモックアップを確認しながら内部ロジックを推測し，実装内容に関する注釈文をモックアップ内に埋め込む（2）．これにより各メンバ間で成果物イメージの具体化・共有化が図られる．両作業を終えたのち，利用者が「コンバート」ボタンを押すことにより，UI パーツと画面遷移に関する処理のみで構成された「プロトタイプ Ver.0」が出力される（3）．「プロトタイプ Ver.0」の実態は UI パーツおよび画面の遷移に関する処理が記述されたメソッド群であり，モックアップ作成の際に（2）で記述された注釈文はメソッド内のコメント文として挿入される．開発過程（5）にてユーザはコメント文に記述された内容（例えばデータフローやコンテンツ制御のアルゴリズムなど）を確認しつつ

「プロトタイプ Ver.0」のメソッド内のプログラムに変更・拡張を施していくことにより，当初のイメージに忠実な成果物が生み出されることを期待する．

5.2.1　Cacoo について

図 5.1　Cacoo での作業画面

本機能は，Nulab 社のデザインツールである Cacoo[27] の機能をベースとしている（図 5.1）．Cacoo では，アプリケーションのワイヤフレーム，フローチャートや UML などを設計するためのテンプレートがサポートされている．利用者はまず最初にダイヤグラムと称されるアートボードを作成したのち，ダイアグラム上にてステンシルと呼ばれる図形の群を用い，アプリケーションの開発に必要な資料を作成する．2015 年 1 月時点においては，iPhone，iPad，Android アプリケーションの基本的な UI パーツ（Label や Button など）がステンシルテンプレートとして利用可能であり（図 5.2），スマートフォン向けアプリケーションを開発する上では本物に近いインタフェースを即興的にデザインできる．ただし，Cacoo でのデザインが可能なのは「アプリケーションの見た目の部分」に限定され，第 3 章にて紹

図 5.2　Cacoo のステンシルテンプレート

介した既存のモックアップツールと同様，作成した図は飽くまでも「アプリケーション開発における参考資料」という扱いである．Cacoo は Web サービスとして提供されており，Web ブラウザを介して利用できるためインストールの手間を必要としない．ダイアグラムの作成者は自分のダイヤグラム上に他のユーザを招待することができ，一つのダイヤグラムをリアルタイムに共同編集できるほか，共同編集者同士でチャットをする機能も提供されている．

5.2.2　モックアップのソースコード変換

Cacoo で作成されたダイヤグラムの情報は，開発者 API を介することで XML データとして取得できる．本機能は，取得した XML データからモックアップの構造を解析することにより，アプリケーションのソースコードを自動生成する．2015 年 2 月現在，本機能は Xcode の Objective-C 言語に対応しており，Cacoo で作成したモックアップを iOS アプリケーションとして書き出すことが可能である．

■ダイアグラムシートの変換　Cacoo のダイヤグラム上では複数の「シート」を作成することが可能である．モックアップをソースコードに変換する際には，1 枚のシート毎に設計されたモックアップがアプリケーション上の 1 画面として扱われる．シートのソースコード変換例を図 5.3 に示す．2 画面で構成されるアプリケーションを開発する場合においては，シートを 2 枚作成する必要がある．シートに名付けられた名前はソースコード変換時にクラス名として扱われ，Objective-C 言語における画面管理クラス「UIViewController」を継承したカスタムクラスとなる．1 枚のシートあたりにクラスインタフェース部分を定義する「.h」ファイルと実装部分を定義する「.m」ファイルがそれぞれ生成されるほか，全体の GUI インタフェースを定義する「.storyboard」ファイル内にモックアップの画面情報が反映される．これらのファイル構成は，Objective-C 言語を用いた iOS アプリケーション開発においてごく一般的である．

■ステンシルの変換　ステンシルテンプレートを用いてモックアップを作成する際，本機能を活用する上で，まずモックアップ画面の土台となるステンシルを設置する必要がある．図 5.4 に土台となるステンシルを示す．iPhone ステンシルもしくは Rounded Rectangle ステンシルをシート上に設置することで，アプリケーションの画面領域を指定することができる．この作業後，画面領域外に設置されたステンシルやその他オブジェクトはソースコード変換時において「無効オブジェクト」もしくは「注釈オブジェクト」として扱われる．画面領域内に設置されたオブジェクトのうち，ソースコード変換対象となるのは図 5.5 のオブジェクト群である．モックアップの解析過程において，これらのオブジェクトはいくつかの属性に再分類されたのち．Objective-C 上のクラスオブジェクトとして「.storyboard」ファイル内に挿入される（図 5.6）．なお Image オブジェクトと Label オブジェクトは，後述の「注釈内容の解析」過程にて特定の条件が満たされた際に Button オブジェクト扱いとなる場合がある．

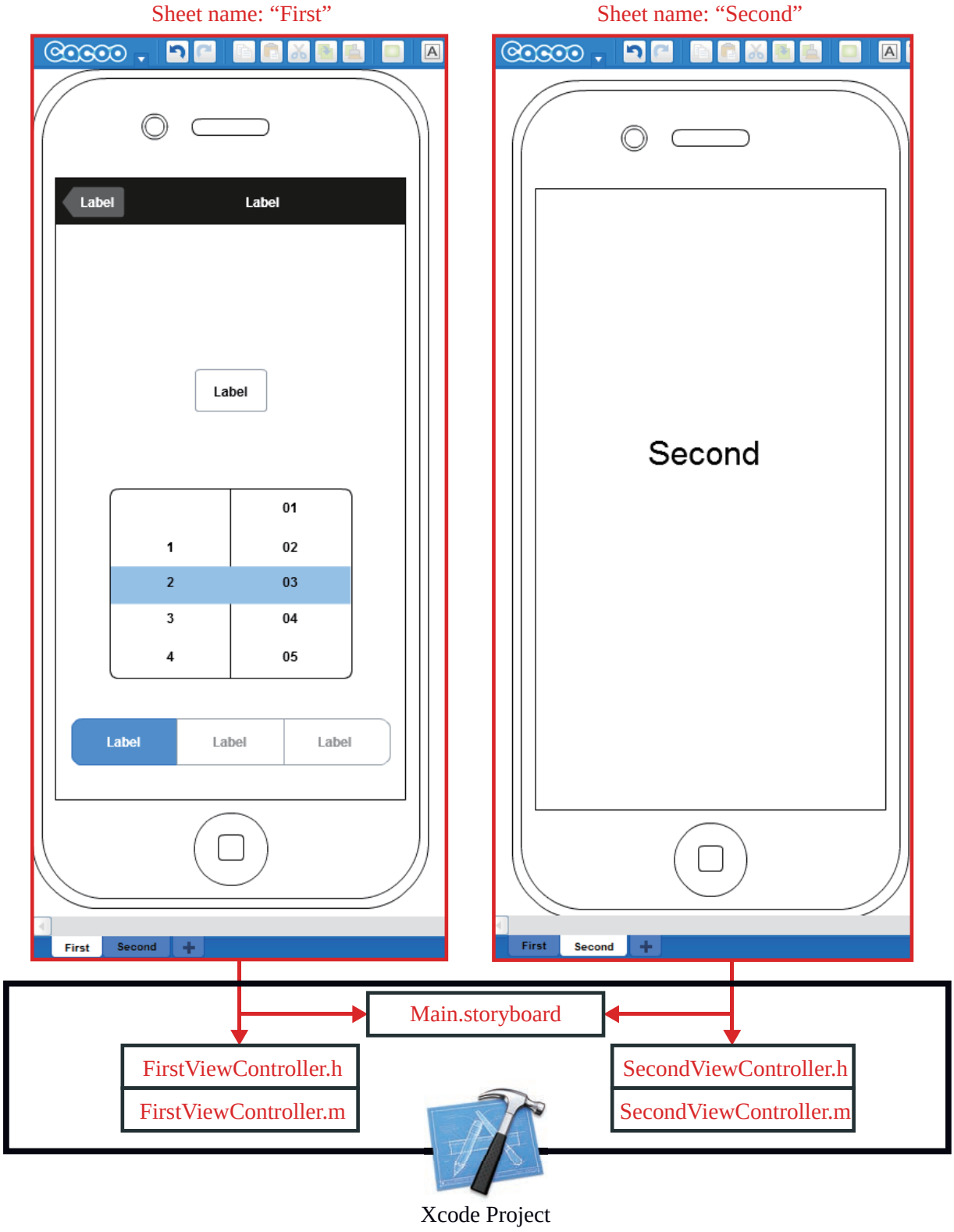

図 5.3　ダイアグラムシートのソースコード化

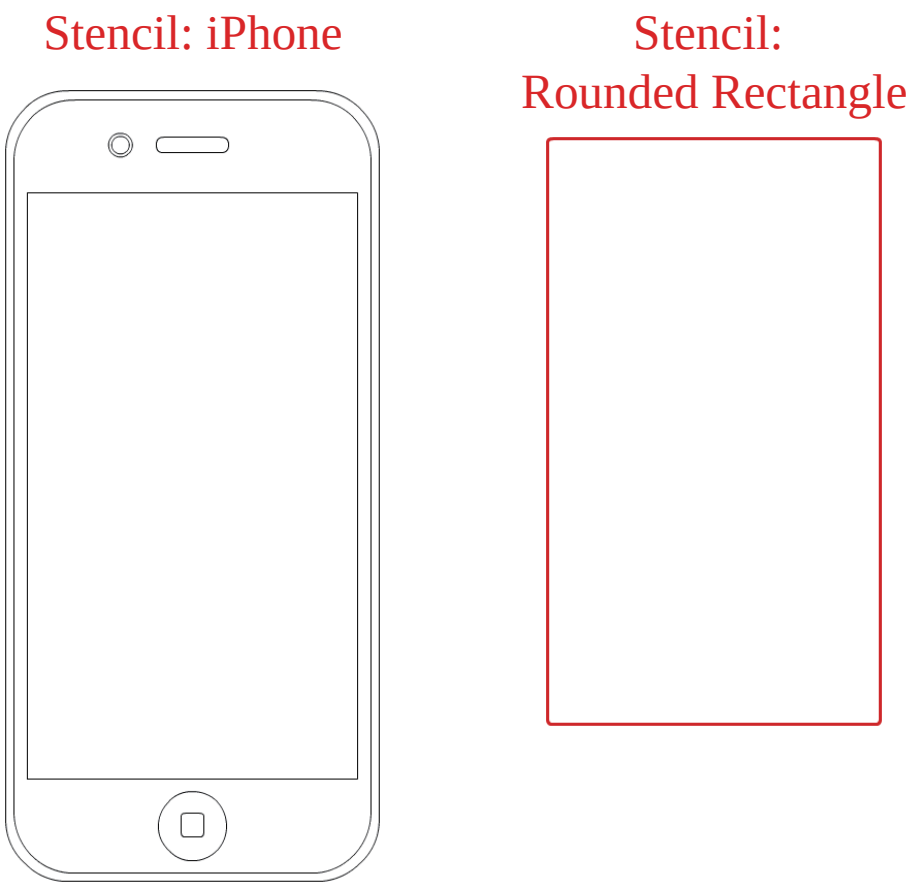

図 5.4　モックアップ画面の土台となるステンシル

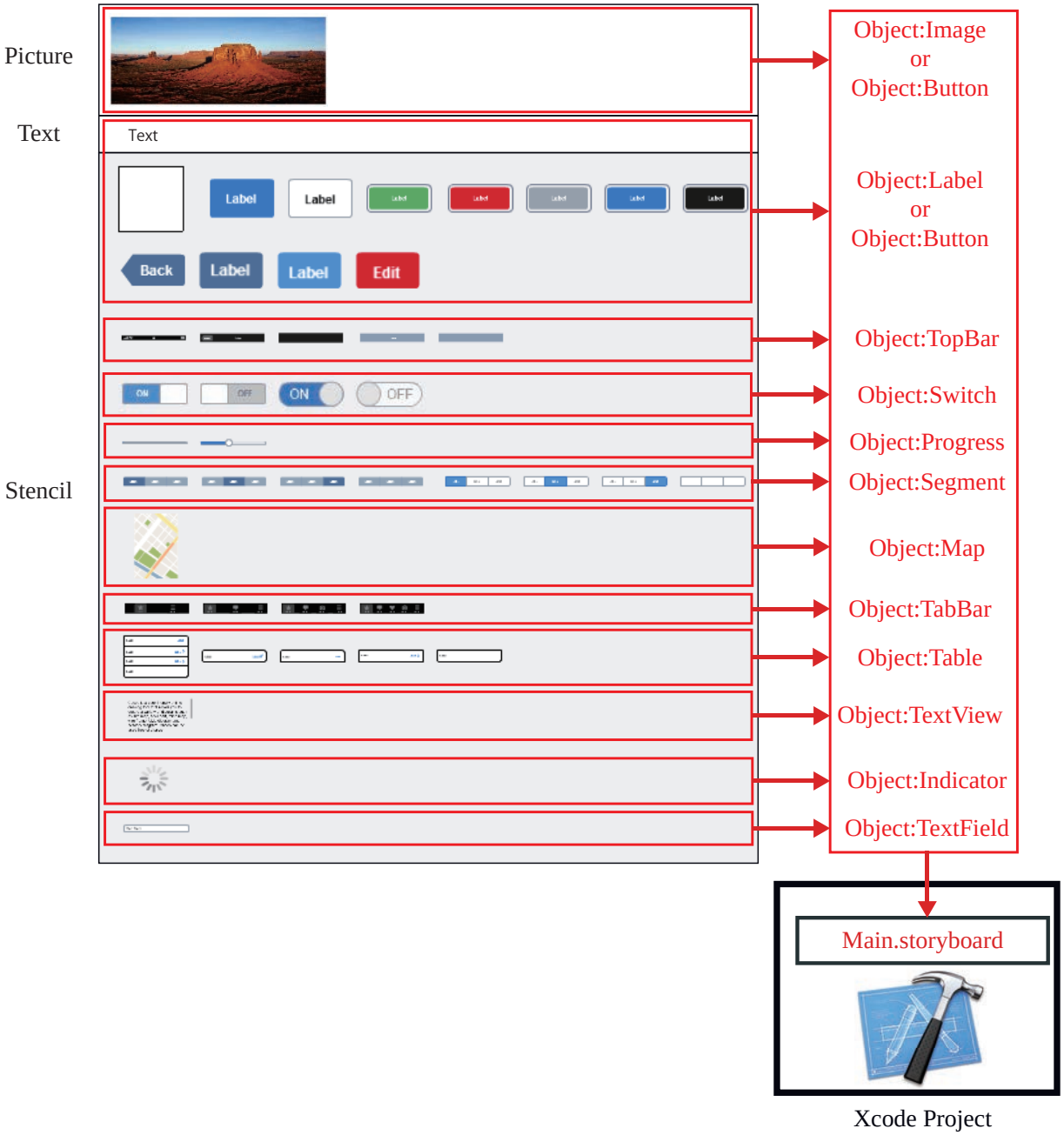

図 5.5　Cacoo オブジェクトの変換対応表

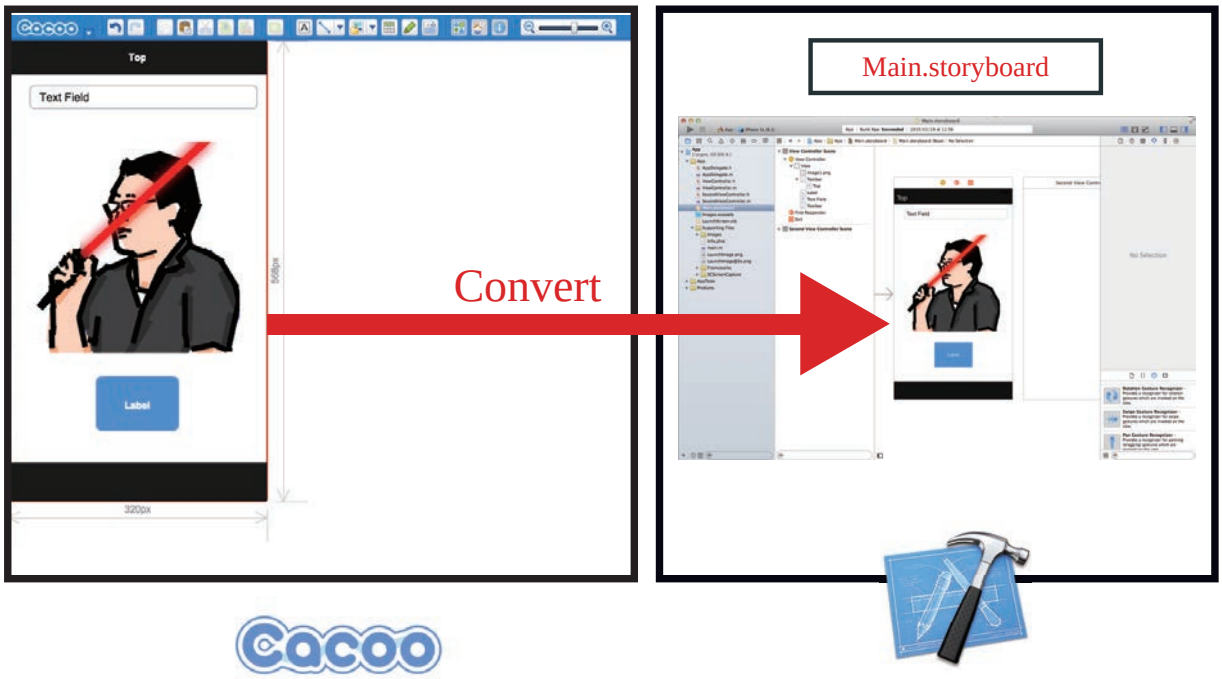

図 5.6　モックアップ画面が反映された Xcode プロジェクト

■注釈内容の解析　土台ステンシル範囲外に設置されたオブジェクトは，開発に関するヒントを示す「注釈オブジェクト」として解釈される．注釈オブジェクト内に埋め込まれたテキストは「注釈文」として解析され，注釈文中の特定の「タグ」に対応した処理を実行する．注釈オブジェクトの設置方法には「モックアップ画面上のオブジェクトに関する注釈」と「内部処理で用いるオブジェクトおよびメソッドに関する注釈」の 2 つのパターンが存在する．

- モックアップ画面上のオブジェクトに関する注釈
  土台ステンシル範囲内のステンシルや画像などのオブジェクトに対して紐づけられ，「タグ」を用いて変数名などを定義する．
  - [outlet_name] タグ
    オブジェクトの変数名を定義する．
    記述例）**[outlet_name]myButton**
  - [action_name] タグ
    オブジェクトがタップされた場合などに呼び出すメソッド名を定義する．
    記述例）**[action_name]pushed**
  - [action_modal] タグ
    オブジェクトがタップされた場合などに画面遷移を実行する．タグの後に記述したシート名が遷移先の画面となる．
    記述例）**[action_modal]Second**
  - [action_dismiss] タグ
    オブジェクトがタップされた場合などに親画面へ再帰する．引数不要．
    記述例）**[action_dismiss]**
  - [comment] タグ
    コメント文として，変数またはメソッド宣言文の真上に挿入される．直前のタグが [outlet_name] だった場合は変数に対するコメント文となり，直前のタグが [action_] の場合はメソッドに対するコメント文となる．
    記述例）**[comment]myButton を押したときに呼ばれるメソッド**

  図 5.5 のオブジェクト群のうち，Image オブジェクトまたは Label オブジェクトに [action_] タグが紐づけられた際には Button オブジェクトとして解釈される．なおオブジェクトに注釈を紐づけるにあたっては，注釈オブジェクトとの関連を示すために「グループ化」の作業が必要となる（図 5.7）．

- 内部処理で用いるオブジェクトおよびメソッドに関する注釈

アルゴリズムの実装などに関連する内部処理のためのオブジェクト・メソッドについて「タグ」を用いて定義する．

– [outlet] タグ
  オブジェクトのクラス名および変数名を定義する．
  記述例）**[outlet]NSArray* myArray**
– [action] タグ
  メソッド名および引数・返り値を定義する．
  記述例）**[action]-(void) myFunction**
– [comment] タグ
  コメント文として，変数またはメソッド宣言文の真上に挿入される．直前のタグが [outlet] だった場合は変数に対するコメント文となり，直前のタグが [action] の場合はメソッドに対するコメント文となる．
  記述例）**[comment]** ○○して××した値を返すメソッドです．
  △△君実装よろしく

注釈内容の解析によってソースコードに挿入された文の例を図 5.8-5.9 に示す．「.h」ファイルにはオブジェクトとメソッドに関する宣言文が挿入され，「.m」ファイルにはメソッドの実装内容が挿入される．また，コメント文の先頭に文字列「TODO:」を挿入することにより，Xcode のファクションメニューからコメント文を参照することができる（図 5.10）．

モックアップと「タグ」を用いることで開発のヒントを示した前例として，Rivero ら [23] の Stractural UI モデルがあるが，Stractural UI モデルは「開発に役立つ参考資料」を作り出すことに留まっているのに対し，本機能においては「急速なプロトタイピング」を可能な限り支援するために，変数やメソッドの宣言文やコメント文を直接ソースコード中に挿入する方法をとっている．

図 5.7　注釈オブジェクトのグループ化

図 5.8　注釈内容の解析によってソースコードに挿入された文（.h）

図 5.9　注釈内容の解析によってソースコードに挿入された文（.m）

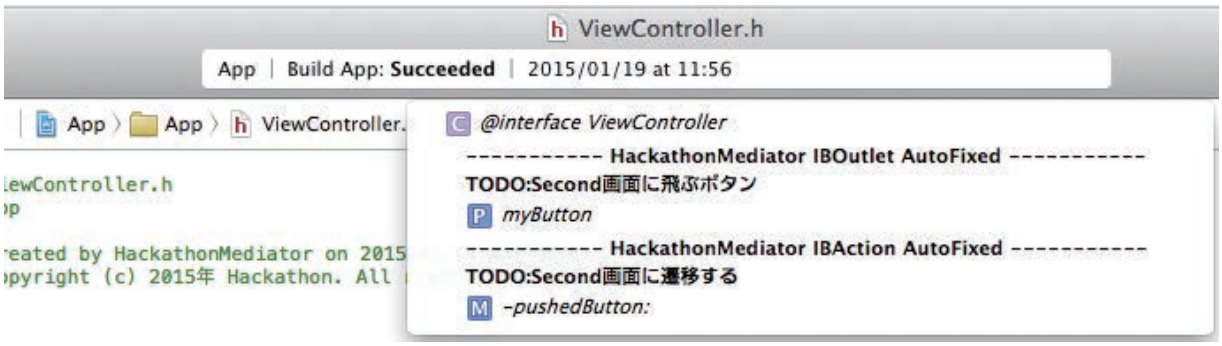

図 5.10　TODO コメント文が表示された Xcode のファンクションメニュー

■シート画像の添付　モックアップのソースコード変換時に，Cacoo で作成した各画面のシート画像が添付される（図 5.11）．この画像は Xcode のプロジェクトナビゲータから参照でき，開発過程のどのようなタイミングにおいても，プロトタイプ Ver.0 作成時のアプリケーション画面を振り返ることができる．

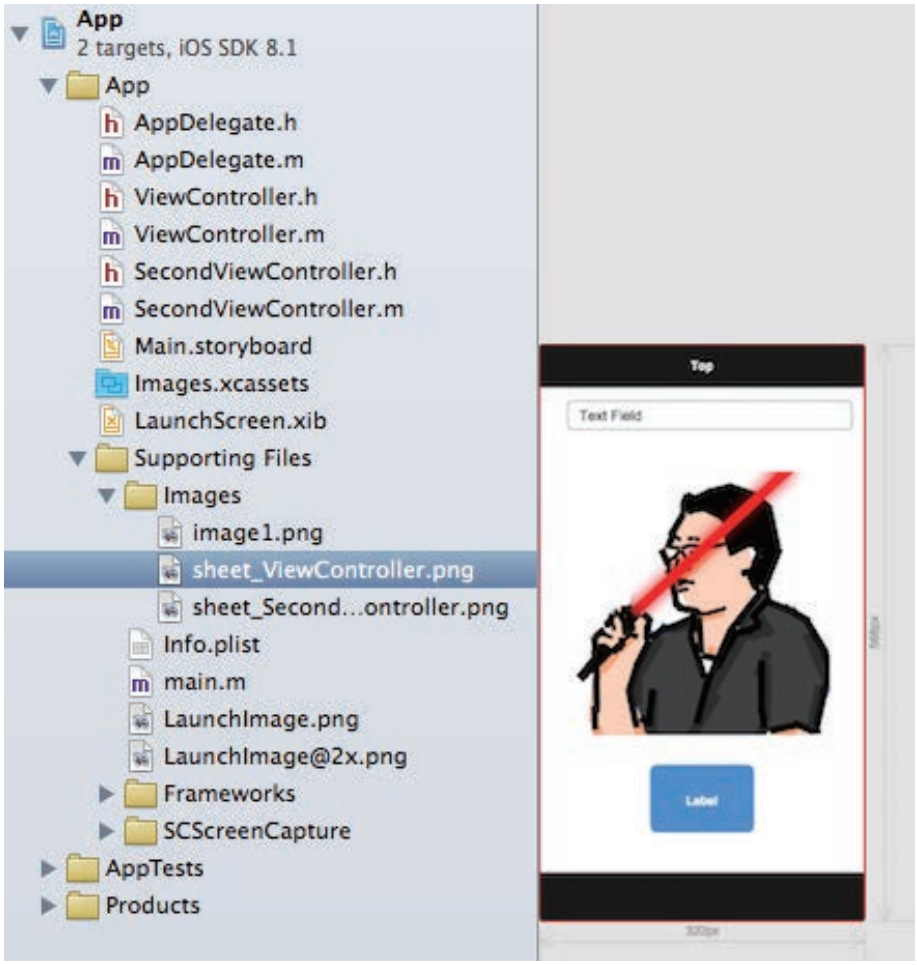

図 5.11　Xcode プロジェクト内に添付されたシート画像

5.2.3　HackathonMediator 用テンプレートシート

本機能を活用する上で，土台ステンシルや注釈オブジェクトなどがあらかじめ配置された「テンプレートシート」を作成した（図 5.12）．テンプレートシート上には，モックアップのソースコード変換を実行する「コンバート」ボタンをはじめ，注釈オブジェクトの記入方法などに関するアナウンスなども設置されている．

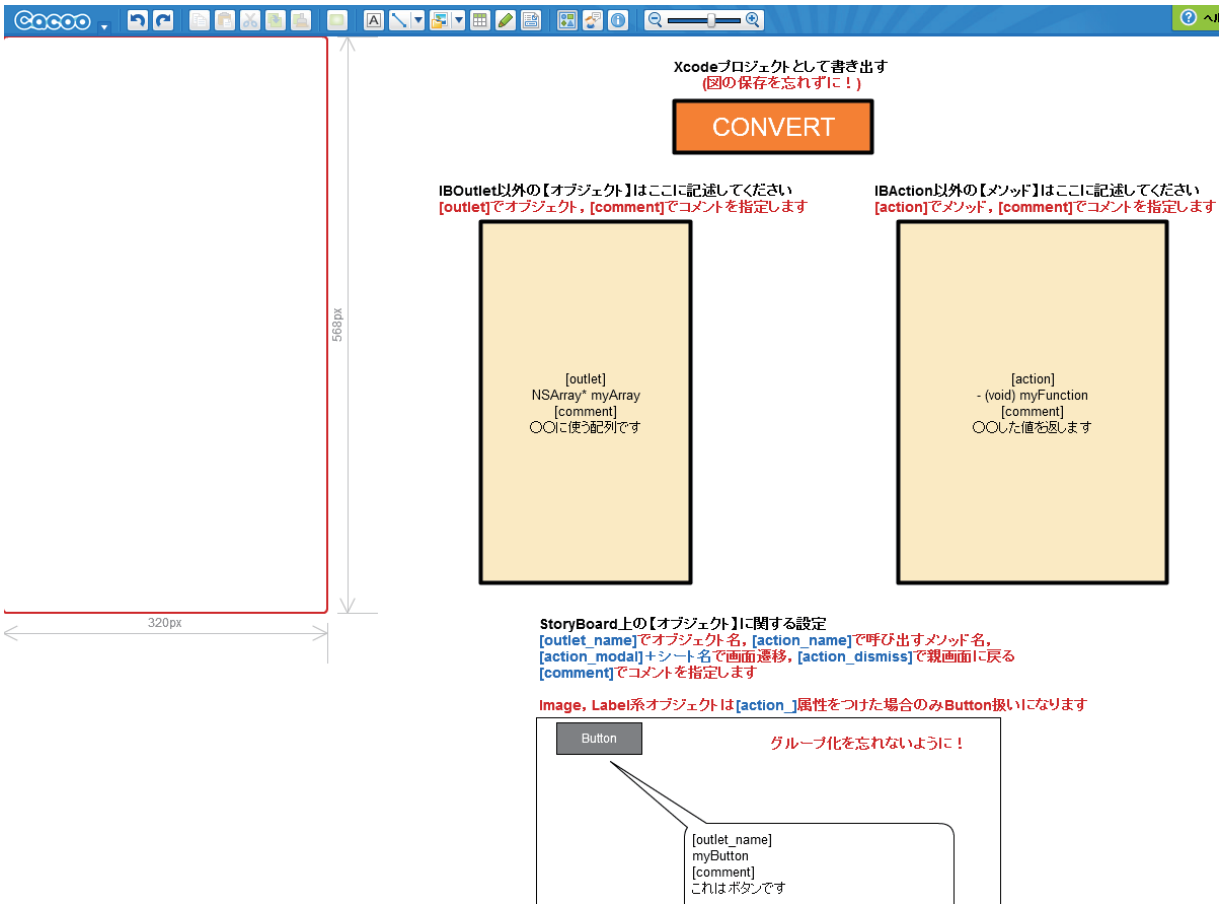

図 5.12　HackathonMediator 用の Cacoo テンプレートシート

5.3　作業進捗共有を兼ねた画面共有機能

本機能では，主にハッカソンの開発過程において，チーム内での作業進捗の共有を支援する．この機能は Windows および Mac OS X の両プラットフォームに向けてそれぞれリリースしており，Windows 版の主な開発言語は C#言語，Mac OS X 版の主な開発言語は Objective-C 言語である．この機能の仕様について図 5.13 に示す．

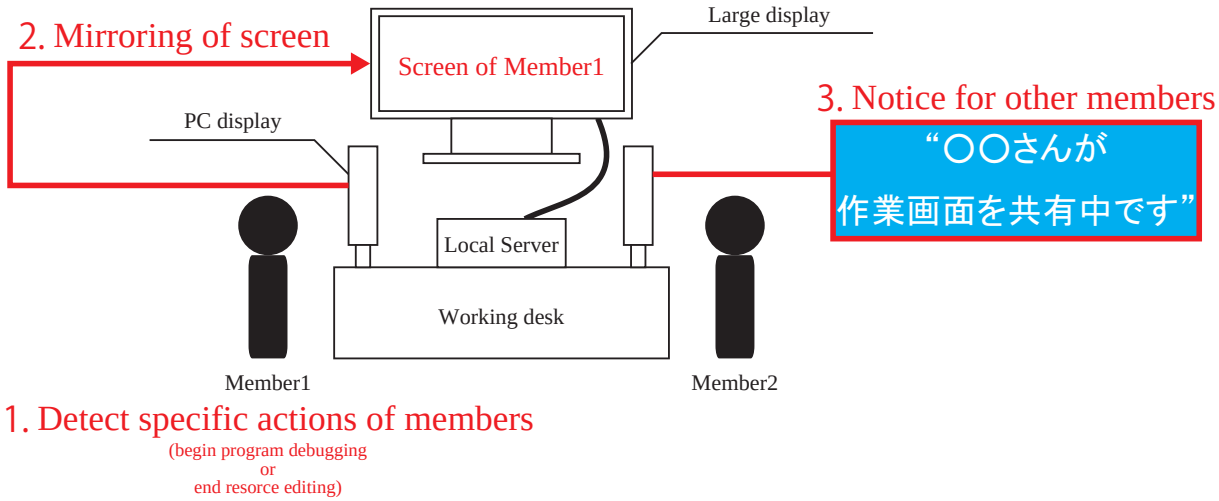

図 5.13　「作業進捗共有を兼ねた画面共有機能」の仕様

1. メンバの特定アクションの検知
   プログラマによるアプリケーションのデバッグ，デザイナやプランナによるリソースの作成・変更などのアクションを検知する．検知方法の詳細については節 5.3.2 の「画面共有のタイミング」にて述べる．
2. 画面共有の実行
   メンバの特定アクションの検知後，該当するメンバの作業画面が大型ディスプレイにミラーリングされる．また，ミラーリングされた画面の上部には，共有したユーザの名前，共有された時間，「ひとことコメント」が表示される（図 5.15）．
3. 他メンバへの通知
   画面共有の実行後，節 5.3.5 で後述する「通知メッセージのレベル設定」機能により，他メンバに対し作業内容の重要度に応じた通知メッセージを発行し，共有された画面を確認するように促す．

本機能の操作画面を図 5.14 に示す．本機能は Mac OS X 版・Window 版においてそれぞれ同様のインタフェースを提供しており，各メンバが自身の作業 PC へインストールして使用する．

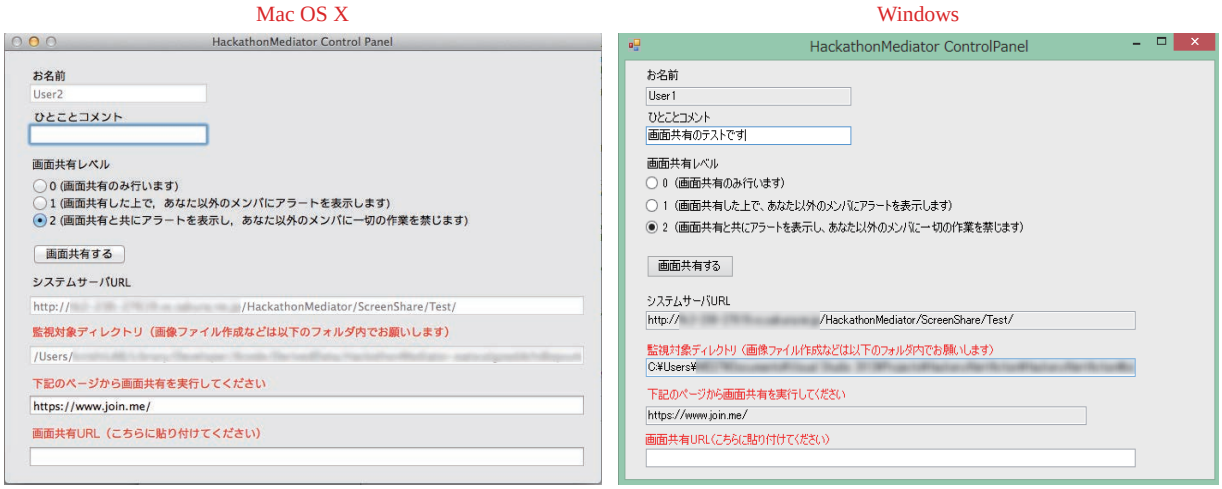

図 5.14　「作業進捗共有を兼ねた画面共有機能」の操作画面

操作画面では画面共有機能に関するいくつかの項目を設定可能であり，各メンバが自身の作業等に合わせてそれぞれ設定する．この操作画面で設定可能な内容は以下である．

- 「お名前」
  チームメンバ名を入力する．
- 「ひとことコメント」
  「○○作業中」など，画面共有時に表示する注釈を入力する．
- 「画面共有レベル」
  従事中の作業の重要度に応じて 3 段階のレベルを設定する．ここで設定したレベルは，節 5.3.5 で後述する「通知メッセージのレベル」に反映される．
- 「画面共有する」
  ボタンを押したメンバの画面共有が実行される．詳しくは「画面共有のタイミング」（節 5.3.2）で述べる．
- 「システムサーバ URL」
  大型ディスプレイに接続されたローカルサーバに「画面共有リクエスト」を発行する URL である．通常は編集しない．
- 「監視対象ディレクトリ」
  リソースの作成や編集を行う際の保存パスを明記する．ここで指定されたパスは節

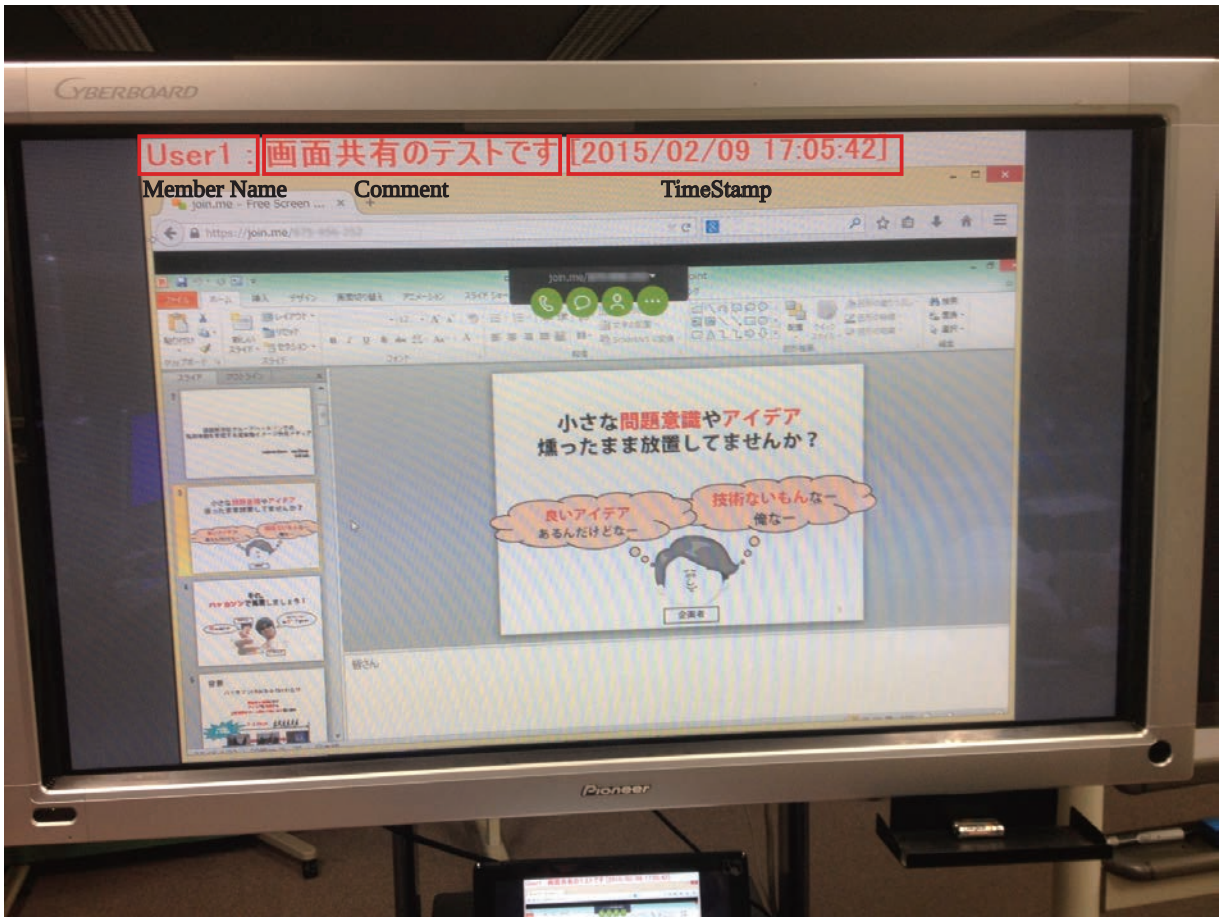

図 5.15　「作業進捗共有を兼ねた画面共有機能」によって共有された画面

5.3.2 で後述する「リソース作成・編集の自動検知」の対象となる．デフォルトでは本機能のインストールパス直下の「workspace」ディレクトリが指定される．

- 「下記のページから画面共有を実行してください」
  本機能の初回起動時に表記 URL にアクセスし，Web 会議用ツール「join.me」（節 5.3.1）を起動状態にする．
- 「画面共有 URL」
  「join.me」を起動した際に発行される専用 URL を張り付ける．

これらの項目以外に，画面共有時においては，共有者の操作画面上に「画面共有停止ボタン」が表示される（図 5.16）．共有者が停止ボタンを押すことにより，画面共有状態が解除される．

図 5.16　「作業進捗共有を兼ねた画面共有機能」の操作画面（画面共有時）

5.3.1　join.me について

本機能は，LogMeIn 社の Web 会議用ツールである join.me[28] の機能をベースとしている．join.me では個人の PC スクリーンを複数人で共有するための機能が提供されており，無償プランにおいても時間制限を受けず長時間の利用が可能なため，Web 会議用ツールの中でも高い人気を誇っている．画面の共有者（配信者）は専用のソフトウェアを起動し，閲覧者用の専用 URL を発行する．閲覧者は Web ブラウザ経由で専用 URL にアクセスすることにより，配信者の共有した画面を閲覧できる．本機能においては各メンバそれぞれが画面共有の配信者の立場となり，大型ディスプレイに接続されたローカルサーバが閲覧者の立場になる．

5.3.2　画面共有のタイミング

画面共有を実行するタイミングにおいては「デバッグの自動検知」，「リソース作成・編集の自動検知」，「共有ボタンの操作」の 3 パターンを提供している．

1. デバッグの自動検知

プログラマがアプリケーションのデバッグ作業を開始した際，自動的に画面共有が行われる．図 5.17 のように，開発途中のアプリケーションが実際に動作する様子をチームで確認できる．具体的な手法としては，メンバの作業 PC 上のプロセス情報を常時監視し，特定の開発ツール上で発生するデバッグプロセスを検出する．ただし，開発ツールのアーキテクチャによってはデバッグプロセスを検出できない場合もあり，Unity などのツールにおいては開発中のプロジェクトに専用パッケージ「HackathonMediator.unitypackage」をインポートすることで対応している．現状においては Xcode，Eclipse，Unity でのデバッグ検知が可能となっている．

2. リソース作成・編集の自動検知

   デザイナやプランナが画像やプレゼンテーション資料などのリソースを新規作成・変更した場合，自動的に画面共有が行われる．これによりイラストツールやプレゼンテーション作成ツールなどの作業画面が共有され，作成状況をチームで確認できる．具体的な手法としては，メンバの作業 PC 上の特定ディレクトリ内を常時監視し，ファイルの作成および変更を検出する．特定ディレクトリ外でのファイル作成・変更に関しては検知が不可能なため，現状においてはユーザに対し，指定のディレクトリ内でのリソース作成を促す必要がある．

3. 共有ボタンの操作

   「デバッグの自動検知」や「リソース作成・編集の自動検知」の例外として，メンバが任意のタイミングで画面共有を行いたい場合には，操作画面上の「画面共有する」ボタンを押すことで画面共有が可能である．

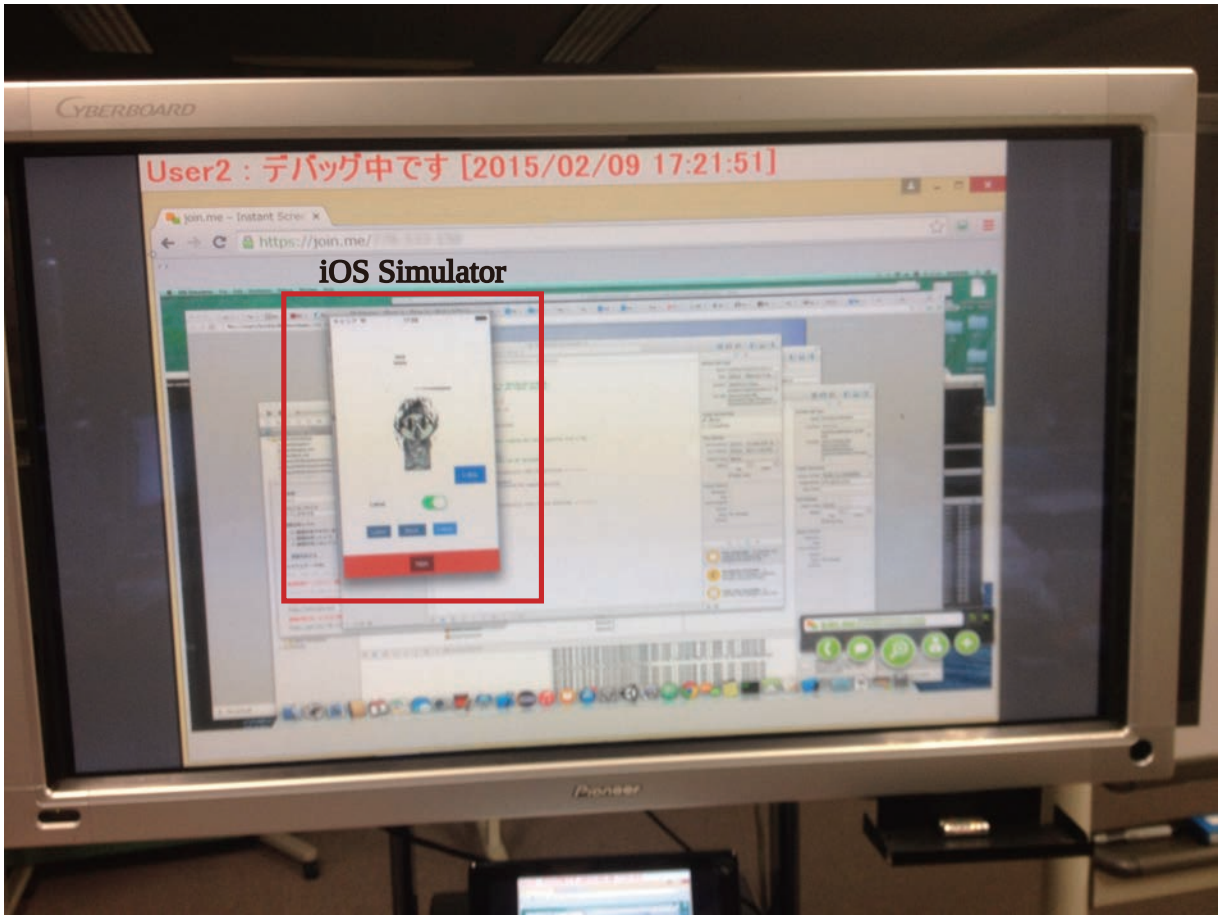

図 5.17　大型ディスプレイ上に共有されたアプリケーションのデバッグ画面

5.3.3　秒読みダイアログの表示

「デバッグの自動検知」または「リソース作成・編集の自動検知」によって画面が共有される場合においては，画面共有を開始する前に「秒読みダイアログ」をミラーリング対象のメンバに対し表示する（図 5.18）．ダイアログは 10 秒間表示され，ダイアログが閉じられた際に画面共有が行われる．メンバはこの間に「画面共有レベル」と「ひとことコメント」を再設定できるほか，ダイアログ上には秒読みを待たずに即共有するための「いますぐ共有」ボタンも設置されている．

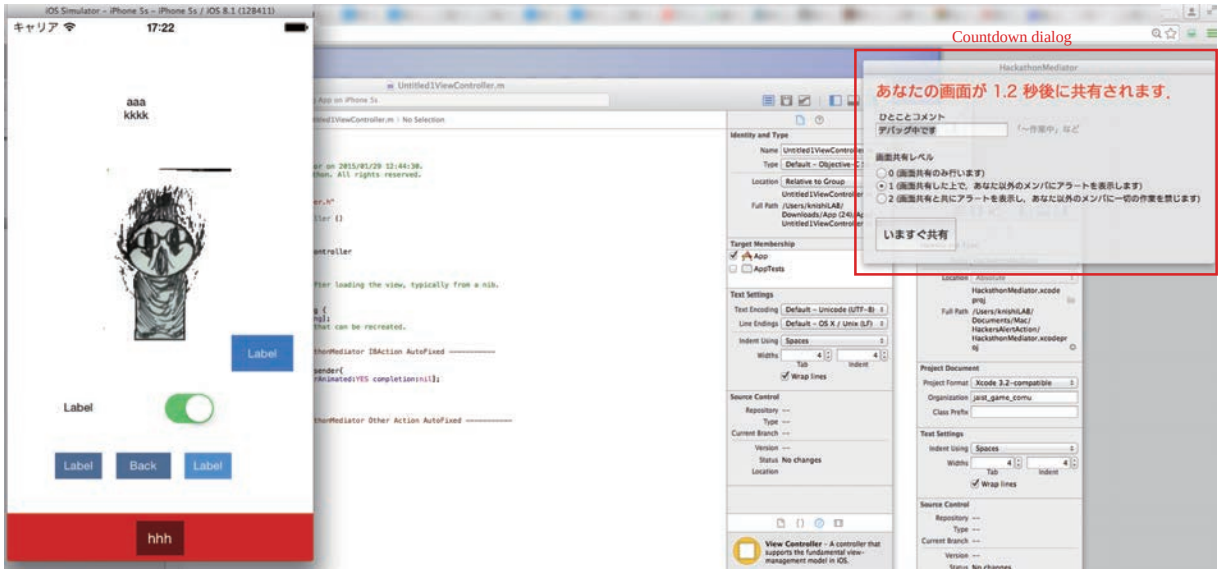

図 5.18　画面共有前の秒読みダイアログ

5.3.4　実機デバッグ時のスクリーン同期機能

スマートフォンアプリケーションをデバッグする際には，シミュレータではなく実機を用いることも想定される．「デバッグの自動検知」において「実機を用いてのデバッグ」を検知した際には，本機能が「実機上のアプリケーション画面」をミラーリング対象者の作業画面上に同期表示する．

シミュレータを用いたデバッグにおいてはアプリケーション画面がプログラマの作業画面上に表示されているため，図 5.17 のような共有が可能となっている．しかし，実機を用いたデバッグにおいてはアプリケーション画面が実機上に表示されるため，「実機上のアプリケーション画面を作業画面上に同期する」機能が必要であった．この機能を実現する上で，

表 5.1　通知メッセージのレベル

Level 0	画面のミラーリングが実行され，効果音が鳴る
Level 1	画面のミラーリングを実行した上で， 効果音と共に他メンバに対し通知メッセージが発行される（図 5.21） 通知メッセージはキー操作によって削除可能である
Level 2	画面のミラーリングを実行した上で， 効果音と共に他メンバに対し通知メッセージが発行され（図 5.22）， ミラーリングしたユーザが「画面共有停止ボタン」で画面共有を取り消すまで 他メンバは一切の作業が不可能になる

「成果物イメージ共有を兼ねたプロトタイプ Ver.0 作成機能」においてソースコードを出力する際に「アプリケーション画面の描画情報」をリアルタイムでローカルサーバにアップロードする機能を組み込んでいる．「実機を用いてのデバッグ」を検知した際，本機能がミラーリング対象メンバの作業画面上に「同期画面」を表示し，ローカルサーバから取得したアプリケーション画面が表示される（図 5.20）．この時，実機画面上でユーザがアプリケーションを操作した際，ユーザによってタップされた画面上の座標を示す赤いマーカーが「同期画面」上に描画される．

5.3.5　通知メッセージの表示

「共有された情報の確認不足」，「他メンバへの関心の希薄化」，「ソロプレイヤー化」などの問題を防止するための方法として，本機能では画面共有時に「通知メッセージ」を表示する機能を提供している．通知メッセージは共有者以外のメンバの作業 PC 上でフルスクリーン表示され，メッセージを削除するまでの間は作業の続行が不可能となる．各ユーザは，自身の画面共有毎に「作業内容の重要度」を 3 段階で指定する．指定された重要度に合わせて，他ユーザに対する通知メッセージの振る舞いが変化する（表 5.1）．作業の重要度に合わせてこれらのレベルを使い分けてもらい，チームに対し気づきや議論の機会を提供する．

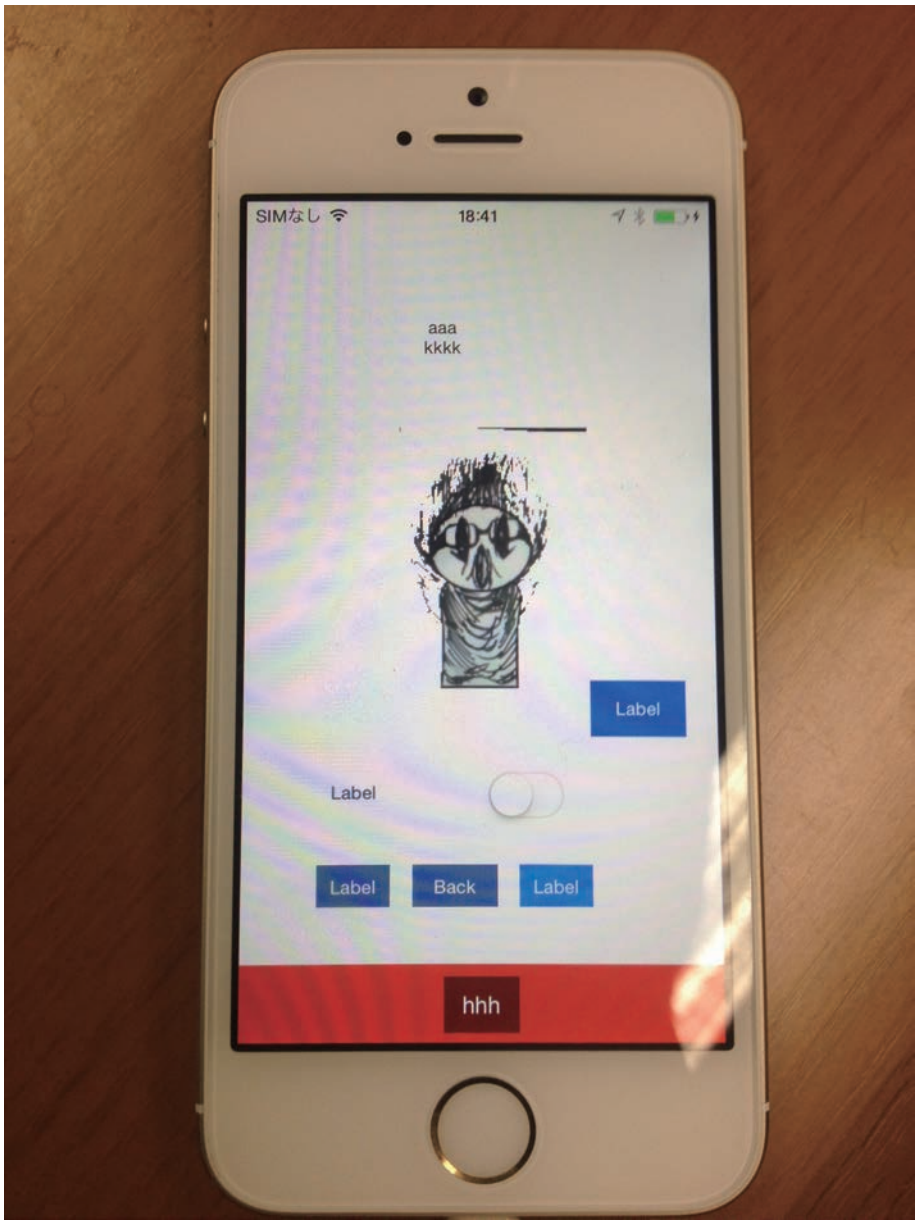

図 5.19　実機（iPhone）を用いたアプリケーションのデバッグ画面

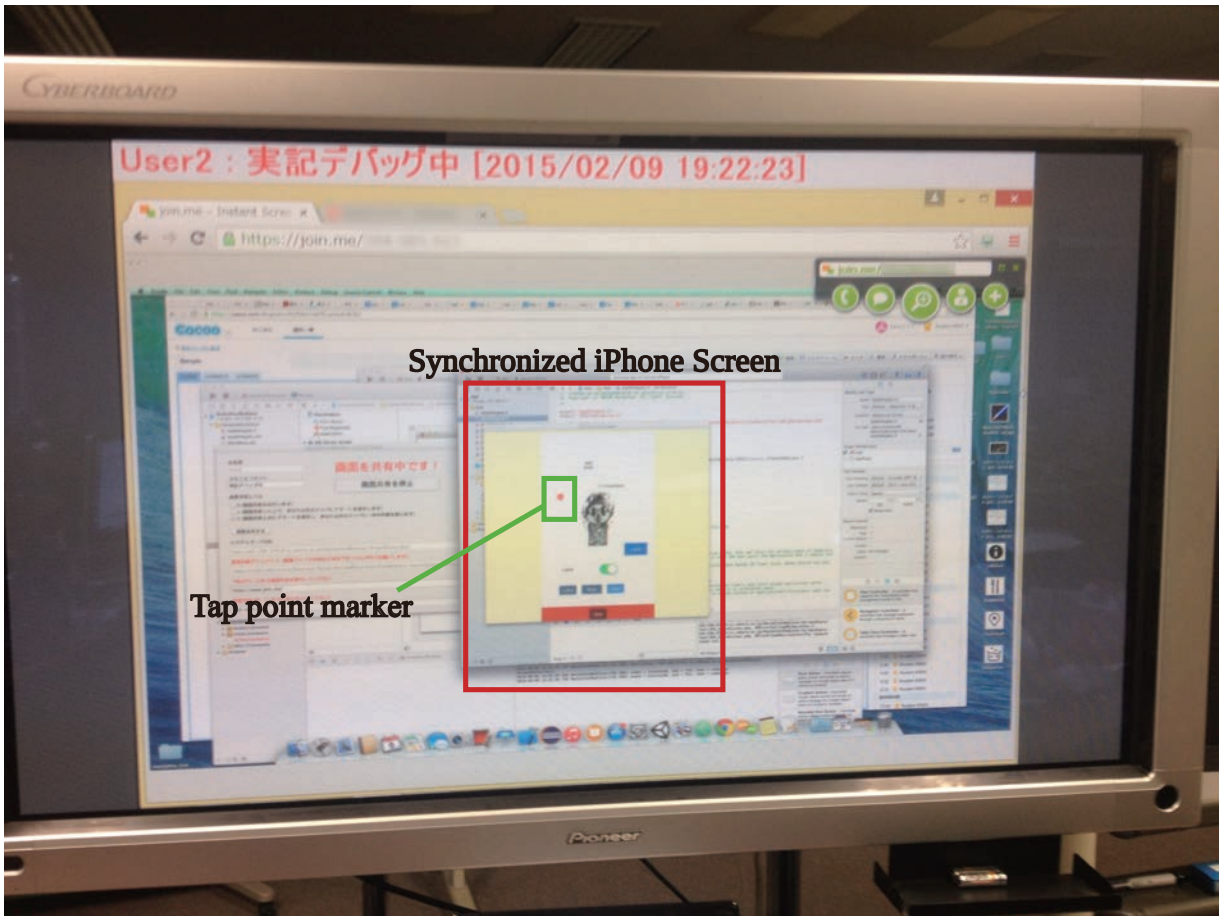

図 5.20　大型ディスプレイ上に共有された「実機（iPhone）を用いたアプリケーションのデバッグ画面」

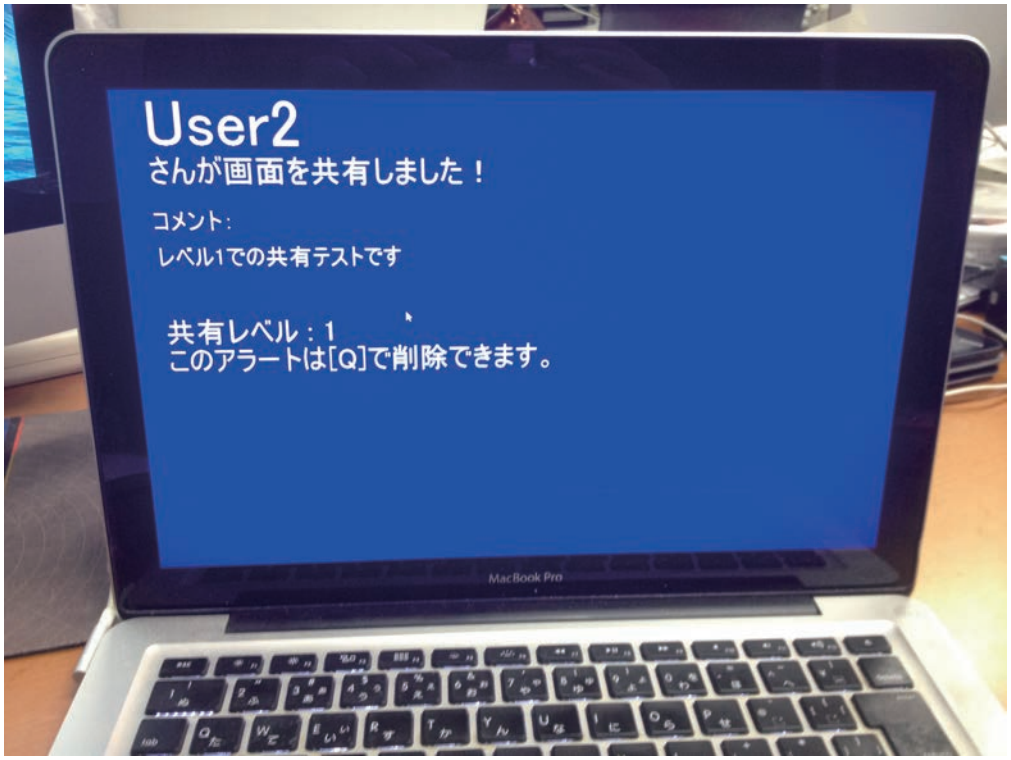

図 5.21　他メンバに対し発行された通知メッセージ（Level1）

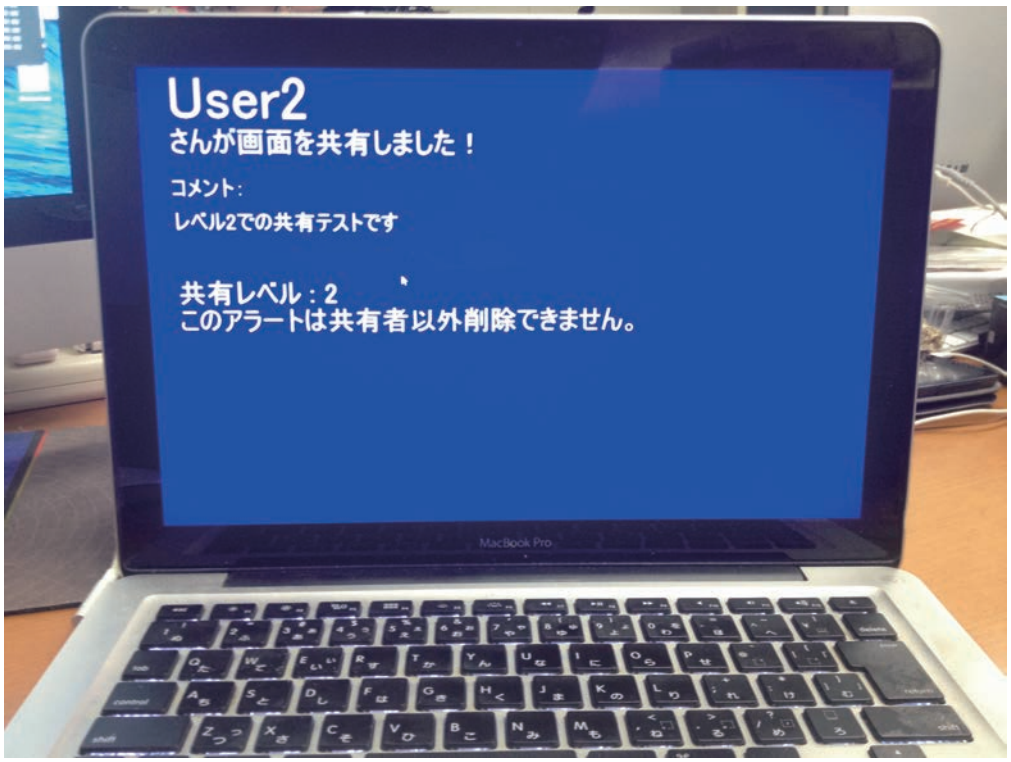

図 5.22　他メンバに対し発行された通知メッセージ（Level2）

5.3.6　画面共有履歴の表示

画面共有が実行されていない時，大型ディスプレイ上には「画面共有履歴」が表示される（図 5.23）．過去に共有された画面を確認できるようにすることで，チームメンバが最近実行した作業内容について把握を促す．

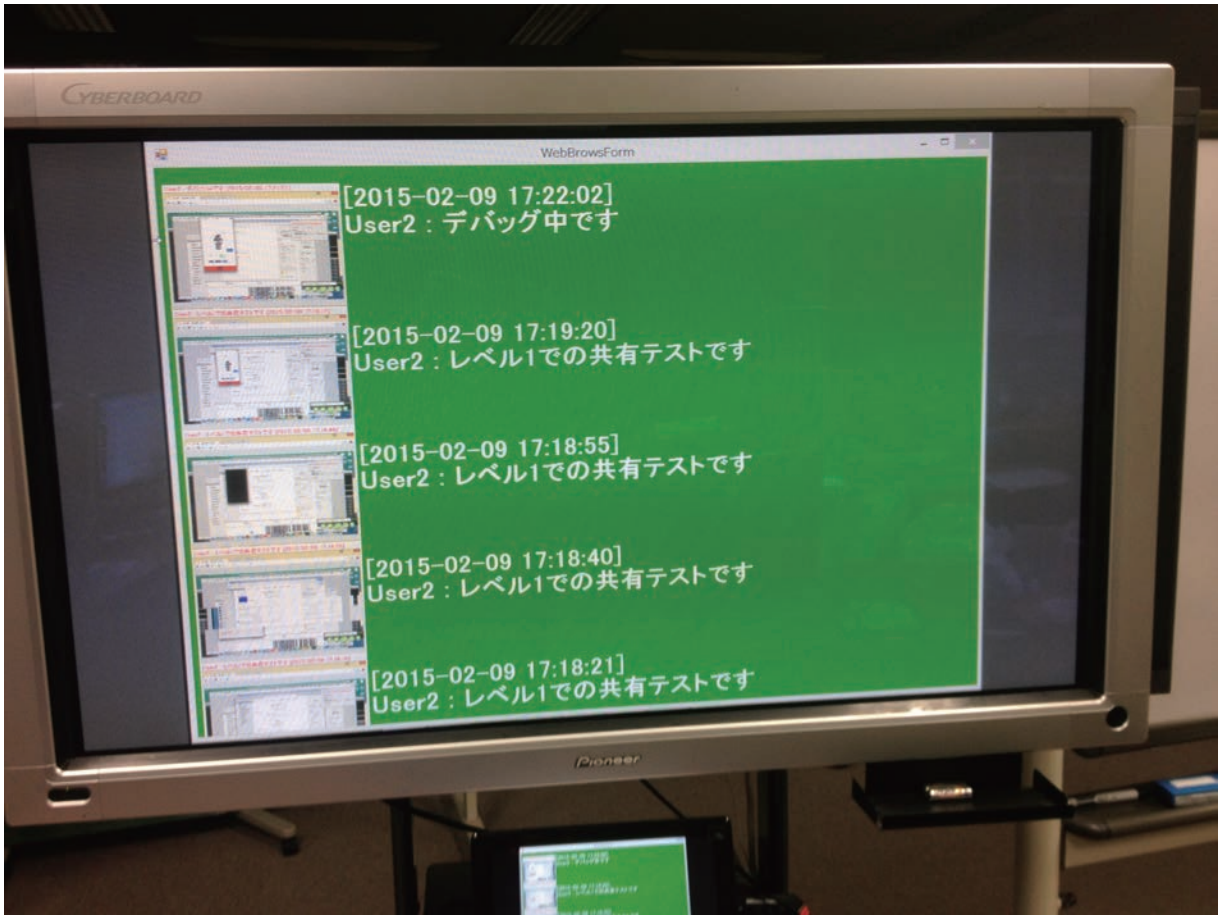

図 5.23　大型ディスプレイ上に表示された画面共有履歴

第 6 章

実験

この章では，第 5 章で述べたハッカソン支援システム「HackathonMediator」を実際のハッカソンに導入し，チームの観察やアンケートで得られた結果について報告する．

第一次実験においては「作業進捗共有を兼ねた画面共有機能」のプロトタイプを導入した実験結果を報告する．第二次実験においては「成果物イメージ共有を兼ねたプロトタイプ Ver.0 作成機能」および「作業進捗共有を兼ねた画面共有機能」の両機能を導入した実験結果を報告する．両実験において，ハッカソンの最中に気になった行動などについては適宜インタビューを行っている．また，ハッカソンの終了後には参加者に対しアンケート調査を実施している．

6.1　第一次実験

第一次実験時点において「成果物イメージ共有を兼ねたプロトタイプ Ver.0 作成機能」は完成しておらず，「作業進捗共有を兼ねた画面共有機能」のプロトタイプを導入した実験結果を報告する．この実験は同機能の設計評価を兼ねたものであった．

「作業進捗共有を兼ねた画面共有機能」のプロトタイプを作成し，実際のハッカソンに導入した．実験は 2014 年 11 月 2-3 日に学内で開催された 30 時間ハッカソンにて実施しており，ハッカソンの開始時に 10 分ほどの説明時間を設け，ハッカソンにおける失敗例と本機能の仕様について参加者に対し説明している．

6.1.1　プロトタイプの仕様

プロトタイプにおける操作画面を図 6.1，プロトタイプによって共有された作業画面を図 6.2 にそれぞれ示す．プロトタイプ時点においては「画面共有のタイミング」と「通知メッ

セージの表示」をサポートしているものの，「画面共有履歴」，「ひとことコメント」，「画面共有の停止」，「秒読みダイアログ」の機能はサポートされていない．「デバッグの自動検知」時や「リソース作成・編集の自動検知」時には即座に画面が共有され，一度共有された画面は，別のメンバによって画面が共有されるまで大型ディスプレイに表示されたままとなる．共有レベル 2 の画面実行時のみ，操作画面（図 6.1）上に「アラートを停止」ボタンが表示され，他メンバに発行した通知メッセージのみを取り消す（画面共有は実行されたままである）ことが可能である．

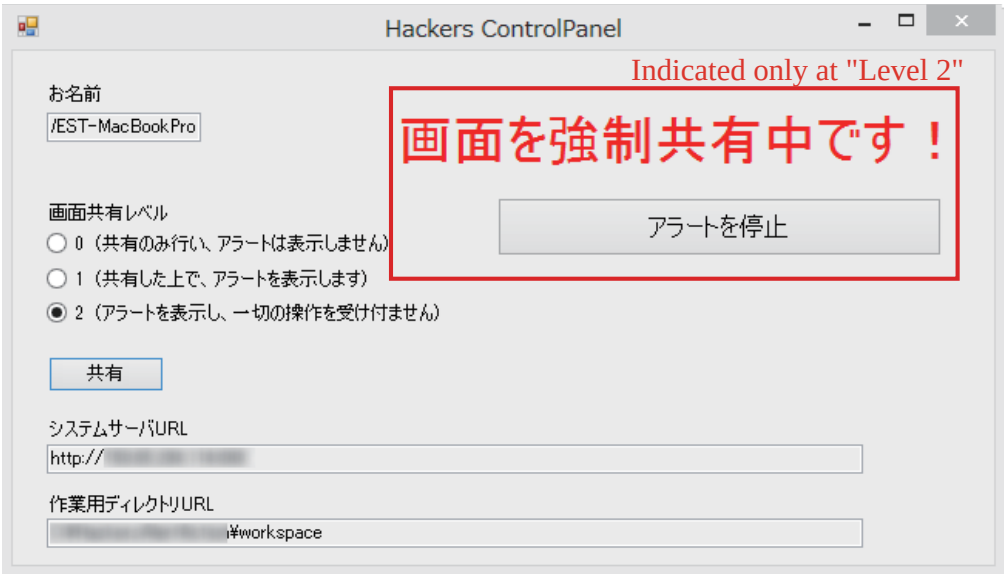

図 6.1　「作業進捗共有を兼ねた画面共有機能」のプロトタイプにおける操作画面

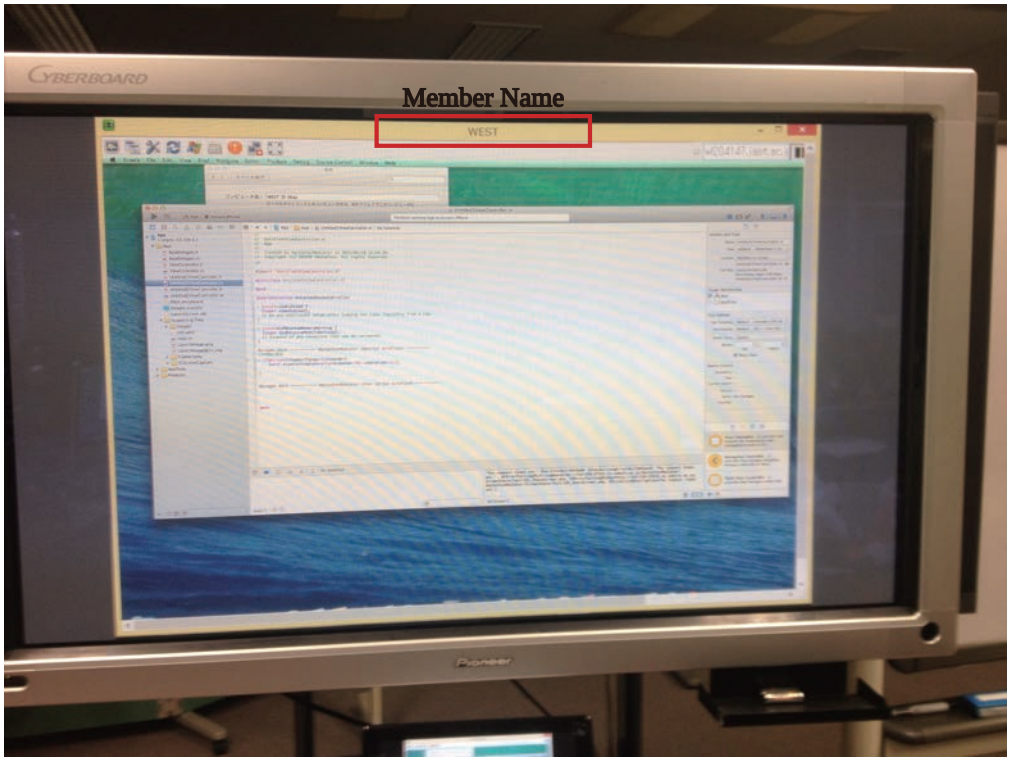

図 6.2　「作業進捗共有を兼ねた画面共有機能」のプロトタイプにおける共有画面

6.1.2　参加者

- チーム A
    1. プログラマ A
    2. プログラマ B
    3. プログラマ C（チームリーダ，プレゼンテータ）
    4. グラフィック・デザイナ
- チーム B
    1. プログラマ A（チームリーダ，プレゼンテータ）
    2. プログラマ B
    3. プログラマ C
    4. サウンド・デザイナ
- チーム
    1. プログラマ A（チームリーダ，プレゼンテータ）
    2. プログラマ B
    3. グラフィック・デザイナ

参加チームは 3 チームであり，チーム A はプログラマ 3 名，デザイナ 1 名の計 4 名，チーム B はプログラマ 3 名，デザイナ 1 名の計 4 名，チーム C はプログラマ 2 名，デザイナ 1 名の計 3 名で構成される．なお，参加者の過去のハッカソン経験について，チーム A のデザイナ，チーム B のプログラマ A，プログラマ B が「複数回経験したことがある」と答えているほか，チーム A のプログラマ C，チーム B のデザイナ，チーム C のプログラマ A が「一度参加したことがある」と答えている．参加者には「“Kami” という文字から連想するアイデアでゲームを作ろう」という共通のテーマが与えられており，全てのチームが Unity を用いてアプリケーションの開発を行った．

6.1.3　チーム A の観察

チーム A においては，結成の初期段階から「コミュニケーションに重きを置いた開発体制の徹底」を宣言しており，各メンバの発言頻度に関して極端な差が見受けられず，チームの連携に関して他チームからも高い評価を受けていた．作業進捗を報告する際には「共有ボタン」を用いた手動での画面共有を積極的に活用する傾向にあり，他メンバに対し「今から共有しますね」と口頭での確認を取った上で「共有レベル 2」での画面共有を行う場面が多

かった．同会場内の他チームのメンバはこのチームを「理想的なチーム体制」と評価した．チーム A のデザイナは「これまで参加したハッカソンの中で一番良いチームワークだった」という感想を述べており，その理由について「企画時点で技術的な課題をほぼ網羅し，ゲームをシンプルなものに設計できていた」ことや「みんなで積極的に意見を出し合う関係が築けていた」ことが示唆された．

6.1.4　チーム B の観察

チーム B においては，プログラマ A とデザイナが積極的に発言する画面が多く見受けられたことに対し，プログラマ B とプログラマ C が発言する機会は少なく見受けられた．このチームは開発を開始する前に「TODO リスト」の作成を行っていた．企画段階にて議論した各作業項目を Post it に一枚ずつ書き起こし，それらは作業机の空きスペースに並べられた．開発過程にて Post it の内容に該当する作業を終える度にその Post it をはがす．チーム B はこれを繰り返すことで「優先順位の誤り」や「待機状態化」などに相当する問題を未然に回避しようとした．しかし，開発終盤において「想定外の作業の発生」が確認されており，この理由について，デザイナから「初期に TODO リストを作成した際，いくつかの作業項目が足りなかったことに気づかなかった」ことが示唆された．また，デザイナの用意した音声素材が本人の意図せぬシチュエーションで使われていたことが指摘されており，「成果物に抱くイメージの差異」，「リソースの未使用」に関連する問題が発生していたことが示唆された．開発過程において，プログラマ B とプログラマ C はアプリケーションの内部ロジックに関係する部分を担当していたことが報告されており，主にプログラム C の作業画面が「デバッグの自動検知」によって頻繁に共有されている．また，チームは「TODO リスト」の作成後，1 時間程度毎に全体での進捗報告の時間を設ける体制を取っており，その際に各メンバは「共有ボタン」を用いて「共有レベル 0」での画面共有を行っていた． このチームにおいては，発話の少なかったプログラマ C のデバッグ画面が高い頻度で共有されており，共有された作業画面を確認した他のメンバが意見や感想を述べるなど，本機能によって発言量の少ないメンバにコミュニケーションの機会をもたらす効用が見られた．

6.1.5　チーム C の観察

チーム C においては，プログラマ A とプログラマ B が積極的に発言する場面が多く見受けられたことに対し，デザイナが発言する機会はあまり確認されなかった．プログラマ A とプログラマ B の関係については，「幼少期からの幼馴染」であることが判明しており，旧知の間柄であるプログラマらとデザイナの間には遠慮の存在が懸念された．開発過程において

は,「リソース作成・編集の自動検知」によりデザイナの作業画面がほぼ常に共有された状態にあり，デザイナの画面共有頻度に対してプログラマの画面共有の頻度は低かった．その理由として，プログラマ A は PC 環境の不具合で本機能の画面共有が実行されなかったことが複数回確認されており，またプログラマ B はハッカソンの最中に本機能を複数回シャットダウンしていたことが判明している．この件についてプログラマ B は,「デザイナの作業画面を見ながら作業したかったので，自分のデバッグ画面に切り替わることを避けたかった」ことを理由として述べている．しかし，デザイナの共有画面がほぼ常に表示されたことで，プログラマがデザイナに意見や感想を述べる場面が多く確認されており，デザイナに対しチーム B のプログラマ C と同様の効用が見られた．なお，開発終盤においてプログラムの統合作業を行っていた際，プログラマ A とプログラマ B の間で口論が発生した．口論の内容は，ソースコード中の変数やメソッドの命名に同一のものを使っていたため，統合時に問題が発生したことにあった．プログラマ A はプログラマ B に対し「命名規則が適当すぎる」と苦言を発した．

6.1.6　アンケート

ハッカソン終了後に各参加者に対しアンケートを実施した．アンケートの内容を以下に示す．

- チームの内省評価
  - 今回の制作にて苦戦した点はありましたか？
  - チーム内では十分な進捗共有がはかられましたか？
- 「作業進捗共有を兼ねた画面共有機能」プロトタイプの評価
  - 機能の使い勝手はどうでしたか？
  - 3 段階の共有レベルの中で一番多用したものはどれでしたか？
  - そのレベルを多用した理由について教えてください
  - この機能の良いと感じた点について教えてください
  - この機能の悪いと感じた点について教えてください
  - この機能に追加すべきだと思う要素はありますか？
  - 画面共有によってディスカッションや発話量は増えましたか？
  - 最初から最後まで機能を活用しましたか？
  - (No の場合) その理由について教えてください

以降にて，アンケートの回答結果を述べる．このアンケートは全ての項目が自由記述形式であり，一部メンバの回答が存在しない項目もあった．

チームの内省評価に関するアンケート結果

ここでは，チームの内省評価に関するアンケートの回答結果について述べる．

■苦戦した点 「今回の制作にて苦戦した点はありましたか？」という質問に対し，各チームは以下のように述べた．チーム A は，チームワーク

- チーム A
  - プログラマ B
    「分担した仕事を統合するのに時間がかかった」
  - プログラマ C
    「企画の立案者になって，作りたいと思うゲームを作れたのはうれしかったが，意見をよく求められたきにキツい面もあった．イメージを伝えるのが難しい」
- チーム B
  - プログラマ A
    「プログラムのステート管理，シーケンスを十分に考えず開発し始めたところ」
  - プログラマ B
    「最後のつめこみでのあせりと自分が書いたところから起きたバグ」
  - プログラマ C
    「Unity の熟練度が低かった．リポジトリの共有」
  - デザイナ
    「完全な個人作業が少なく，個々の作業が全体に影響しやすかった」
- チーム C
  - プログラマ A
    「ほとんどの点」
  - プログラマ B
    「人数が少なく人手が足りなかった」

■進捗共有の有無 「チーム内では十分な進捗共有がはかられましたか？」という質問に対し，各チームは以下のように述べた．

- チーム A

  - プログラマ A
    「十分行えた」
  - プログラマ B
    「十分行えた」
  - プログラマ C
    「十分行えた．こまめに進捗報告することを意識していたのでよくできた」
  - デザイナ
    「十分行えた」
- チーム B
  - プログラマ A
    「細かい処理の流れの確認が不十分だった」
  - プログラマ B
    「十分行えた」
  - プログラマ C
    「十分行えた」
  - デザイナ
    「どちらともいえない」
- チーム C
  - プログラマ A
    「十分とは言い切れないが，それなりに」
  - プログラマ B
    「十分行えた」
  - デザイナ
    「わりと怪しい」

「作業進捗共有を兼ねた画面共有機能」のプロトタイプの評価に関するアンケート結果

ここでは,「作業進捗共有を兼ねた画面共有機能」のプロトタイプの評価に関するアンケートの回答結果について述べる．

■機能の使い勝手 「機能の使い勝手はどうでしたか？」という質問に対し，各チームは以下のように述べた．

- チーム A

- プログラマ A
  「便利だったが，カクつきが気になった」
- プログラマ B
  「便利だったが，カクつきが目立った」
- プログラマ C
  「便利だったが，PC の不具合で使えなくなることがあった」
- デザイナ
  「現状の状態を理解しやすかった」

- チーム B
  - プログラマ A
    「モニタを直接見せずらい相手に見せられ便利だった」
  - プログラマ B
    「画面を見てもらえるのは良かったが，フレームレートが低いのは残念だった」
  - プログラマ C
    「かなり良い．会議の時の大型ディスプレイは必須だと思う」
  - デザイナ
    「あまり有効に感じられなかった」
- チーム C
  - プログラマ A
    「よかった」
  - プログラマ B
    「絵を皆で見るとき便利だった」
  - デザイナ
    「効果音で気づくことができるのでいい．しかし音の煩さがネックになった」

■活用傾向 「最初から最後まで機能を活用しましたか？」,「(No の場合) その理由について教えてください」という質問に対し，各チームは以下のように述べた．

- チーム A
  - プログラマ A
    「しなかった」,「動きの多いゲームであったため，カクつきが気になり，画面を直接見せ合う事もあった」
  - プログラマ B

「最初の方で多用した」
  - プログラマ C
    「自分自身としてはあまり使わなかった」,「慣れていないツールを使うのが苦手」
  - デザイナ
    「最後はあまりしなかった」,「重くなって画面共有がカクついた」
- チーム B
  - プログラマ A
    「1 時間の報告毎に使っていた」
  - プログラマ B
    「(レベル 2 などは) あまりできなかった」,「相手の作業を妨害してしまうと思った」
  - プログラマ C
    「はい」
  - デザイナ
    「いいえ」,「(音を作っていたため) PC での作業が少なかった」
- チーム C
  - プログラマ A
    「最後はあまり使わなかった」,「直接メンバーの横で話していた」
  - プログラマ B
    「しばしば」
  - デザイナ
    「途中から使わなくなった」,「音が煩くて妨げになっていた？」

■多用した共有レベル 「3 段階の共有レベルの中で一番多用したものはどれでしたか？」,「そのレベルを多用した理由について教えてください」という質問に対し，各チームは以下のように述べた.

- チーム A
  - プログラマ A
    「0」,「メンバーの作業の邪魔にならないようにするため」
  - プログラマ B
    「0」,「他のメンバーの作業を強制的に中断させるほど大事なことはなかったし，メンバーも手を休めて確認してくれた」

- プログラマ C
  「0」,「メンバーの作業の邪魔にならないため」
- デザイナ
  「0」,「全員で共有するまでもない場合が多かった」

- チーム B
  - プログラマ A
    「0」,「相手に見せる必要がないと思うことが多かった」
  - プログラマ B
    「0」,「メンバーの作業を妨害したくない．特に画面を見てもらいたいわけではないときが多かった」
  - プログラマ C
    「1」,「みてもらうため．コードレビューや画面レビューなど」
  - デザイナ
    「0」,「(音を作っていたため）共有するような作業が少なかった」
- チーム C
  - プログラマ A
    「0」,「スクリーンに大きく映るし，音も鳴った．完全に作業を止めるレベル 2 や 1 より勝手が良かった」
  - プログラマ B
    「0」,「迷惑をかけないため」
  - デザイナ
    「0」,「一番作業を邪魔しないレベル」

■ディスカッションや発話量への影響 「画面共有によってディスカッションや発話量は増えましたか？」という質問に対し，各チームは以下のように述べた．

- チーム A
  - プログラマ A
    「増えた」
  - プログラマ B
    「とても増えた」
  - プログラマ C
    「増えた」

- デザイナ
  「増えた」
- チーム B
  - プログラマ A
    「増えた」
  - プログラマ B
    「比較元がないが，発話量は増えやすくなるのではと思った」
  - プログラマ C
    「増えた」
  - デザイナ
    「増えたかもしれない」
- チーム C
  - プログラマ A
    「少し増えた」
  - プログラマ B
    「絵に関して増えた」
  - デザイナ
    「わりと怪しい」

■機能の活用メリット 「この機能の良いと感じた点について教えてください」,「この機能の悪いと感じた点について教えてください」という質問に対し，各チームは以下のように述べた.

- チーム A
  - プログラマ B
    良い点：「画面共有は便利」
    悪い点：「レスポンスが良くない．たまにエラーがある」
  - プログラマ C
    悪い点：「音が耳障り」
  - デザイナ
    悪い点：「再生が遅れたりしていた」
- チーム B
  - プログラマ A

良い点：「デバッグ時に自動で相手に共有できることは便利」
悪い点：「音がわずらわしい．音も共有したかった．フレームレートが低く，細かい動きが伝わらない」

- プログラマ B
  良い点：「パソコンを見なくても示すことができる」悪い点：「必要がないときも音が鳴り続く，画面が共有される」
- プログラマ C
  悪い点：「Chatwork，idobata 等と連携して欲しい」
- デザイナ
  良い点：「他の人の作業をすぐ見れる点」
  悪い点：「画面上で見せる範囲の制御が出来ない」

- チーム C
  - プログラマ A
    良い点：「メンバーに注目して欲しいタイミングで注目させられるのが良かった」
  - プログラマ B
    悪い点：「画面の動きが不安定だった」
  - デザイナ
    良い点：「通知メッセージを用いた共有」
    悪い点：「通知メッセージが返って作業の妨げになっていそう」

6.2　第二次実験

「成果物イメージ共有を兼ねたプロトタイプ Ver.0 作成機能」および「作業進捗共有を兼ねた画面共有機能」を導入した．「成果物イメージ共有を兼ねたプロトタイプ Ver.0 作成機能」を導入するにあたり，参加者には「Xcode を活用した iOS アプリケーションの開発経験があること」を求めた．本学内において条件に該当する参加者を確保することは難しく，同実験は 2015 年 1 月 30 日に九州産業大学情報科学部で開催された 10 時間ハッカソン「9389」にて実施した．なお，このハッカソンにおいては「学生生活に役立つ iPhone アプリケーションを作る」ことが共通のテーマとして設定されている．ハッカソンの開始時には 15 分ほどの説明時間を設けており，ハッカソンにおける失敗例と HackathonMediator の仕様について参加者に説明を実施した．

6.2.1　参加者

- チーム A
  1. プログラマ A（チームリーダ，プレゼンテータ）
  2. プログラマ B
  3. プログラマ C（プレゼンテータ）
  4. デザイナ
- チーム B
  1. プログラマ A
  2. プログラマ B
  3. プログラマ C
  4. デザイナ（プレゼンテータ）

参加チームは 2 チームであり，チーム A，B 共にプログラマ 3 名，デザイナ 1 名の計 4 名で構成される．参加者の全員がプログラミング経験者であり，チーム A のプログラマ A とプログラマ C，チーム B のプログラマ A とプログラマ B は Xcode の活用歴が半年以上である．なお，参加者の過去のハッカソン経験については，全員が「経験したことがない」と答えている．また，チーム B においては明確なチームリーダが存在しなかった．

6.2.2　チーム A の観察

企画段階での議論にて，チーム A が作成したスケッチを図 6.3 に示す．チーム A はこのスケッチを記述しつつ，プレゼンテーションで強調するコンセプトやアプリケーションの機能に関して 1 時間程度の議論を行った．チーム A の企画内容は「エナジードリンクを飲むたびにアプリのボタンを押してポイントを溜め，一定のポイントが溜まる度にレベルが上がっていき，そのレベルをみんなで競うアプリ」だった．議論の際には冗談などのやり取りが頻繁に交えられており，チームの仲の良さなどがうかがえた．

モックアップの作成

企画終了時にて，チーム A が作成した Cacoo のモックアップを図 6.4 に示す．チーム A は 1 つのモックアップ画面を作成しており，注釈オブジェクト内で内部処理に関わる変数およびメソッドが複数定義されていた．

開発過程での行動

チーム A においては，各メンバが積極的に発言する場面が多く見受けられている．図 6.5 に画面共有履歴の一部を示す．「作業進捗共有を兼ねた画面共有機能」の「デバッグの自動検知」により，プログラマ C の画面が高い頻度で共有されていた．時点で共有頻度が高かったのはプログラマ B の画面であり，主にプレゼンテーションの資料作成時の「リソース作成・編集の自動検知」によるものであった．プログラマ A はサーバサイドの開発を主に担当しており，Xcode を用いた開発にほぼ従事していなかったため，「デバッグの自動検知」による共有機会は少なかったが，「共有ボタン」を押すことでサーバサイドのソースコードを共有するなどの行動が見られた．デザイナはイラストツールを用いて画像を作成していたが，このイラストツールは自身の PC ではなくリモートマシンを経由して活用されており，「リソース作成・編集の自動検知」は作動しなかった．デザイナもプログラマ A と同様に「共有ボタン」を用いての共有を行っていた．

完成したアプリケーション

チーム A が最終的に完成させたアプリケーションの状態を図 6.6 に示す．チーム A はプロトタイプ Ver.0 作成時においてアプリケーションの画面の数を 1 つに設定しており，最終的に完成したアプリケーションも 2 つの画面で構成されたものだったが，うち 1 つの画面に関しては遷移が不可能な状態となっていた．

ソースコードの状態

チーム A が最終的に完成させたアプリケーションのソースコードの一部を図 6.7 に示す．チーム A は，「成果物イメージ共有を兼ねたプロトタイプ Ver.0 作成機能」のソースコード変換機能を使用し，モックアップから変換されたソースコードを用いて開発を行っている．アプリケーションのソースコード中において，いくつかのオブジェクト変数とメソッドに関する宣言文およびコメント文が，ソースコード変換機能によって挿入されていることを確認できた．ソースコード変換機能によって挿入されたメソッドは 7 つであり，その内 6 つのメソッドの内容がメンバによって実装されていた．これらのメソッドの実装により，スケッチ作成時に議論された根幹機能が実現された状態となっていたことを確認した．また，新たな拡張機能として，「別バージョンの画面を表示する機能」や「効果音を再生する機能」などの実装が見られた．ソースコード変換時に挿入されたコメント文以外に，チームが新たに記述したコメント文は発見されなかった．

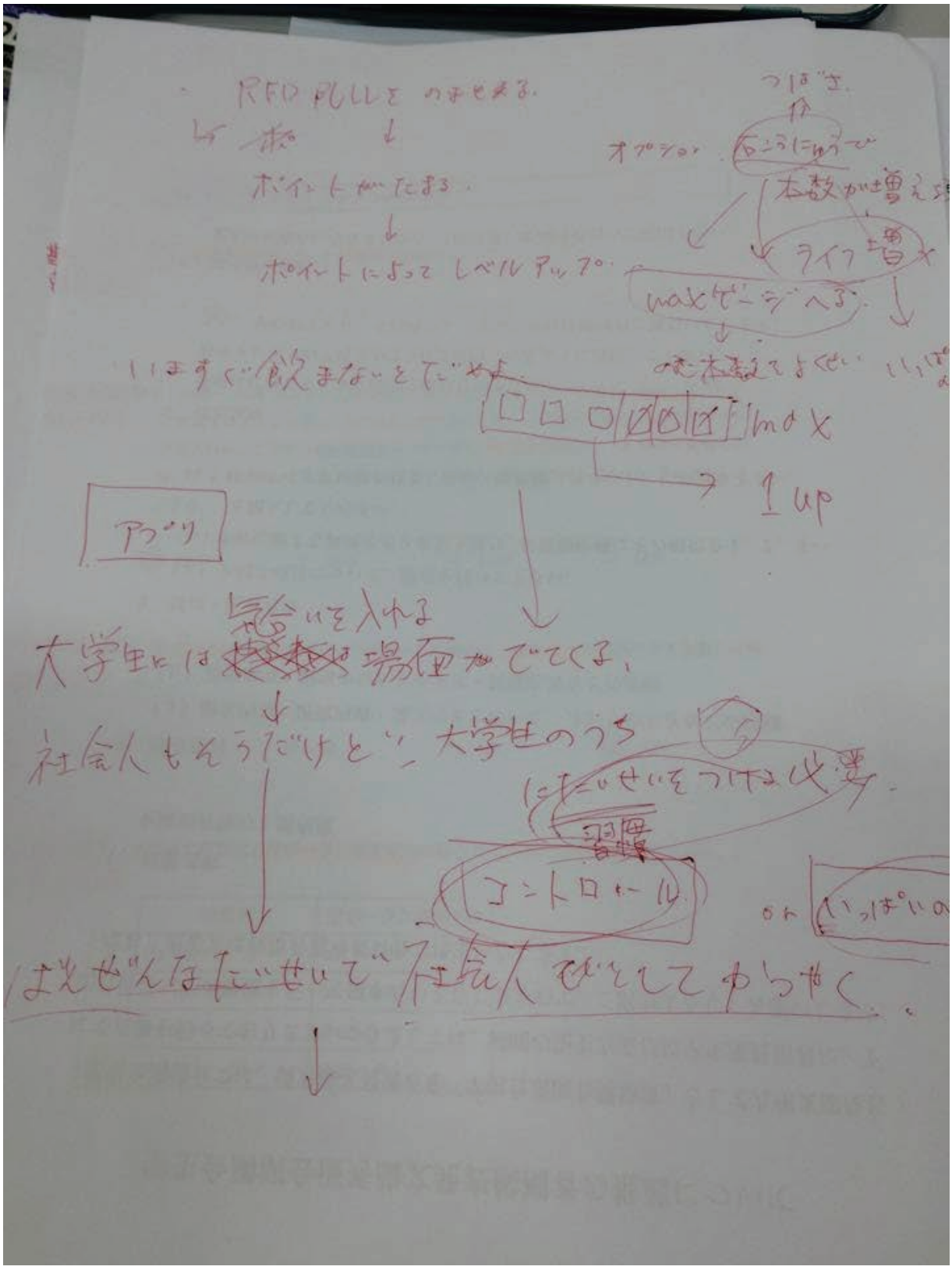

図 6.3　企画段階にチーム A が作成したスケッチ

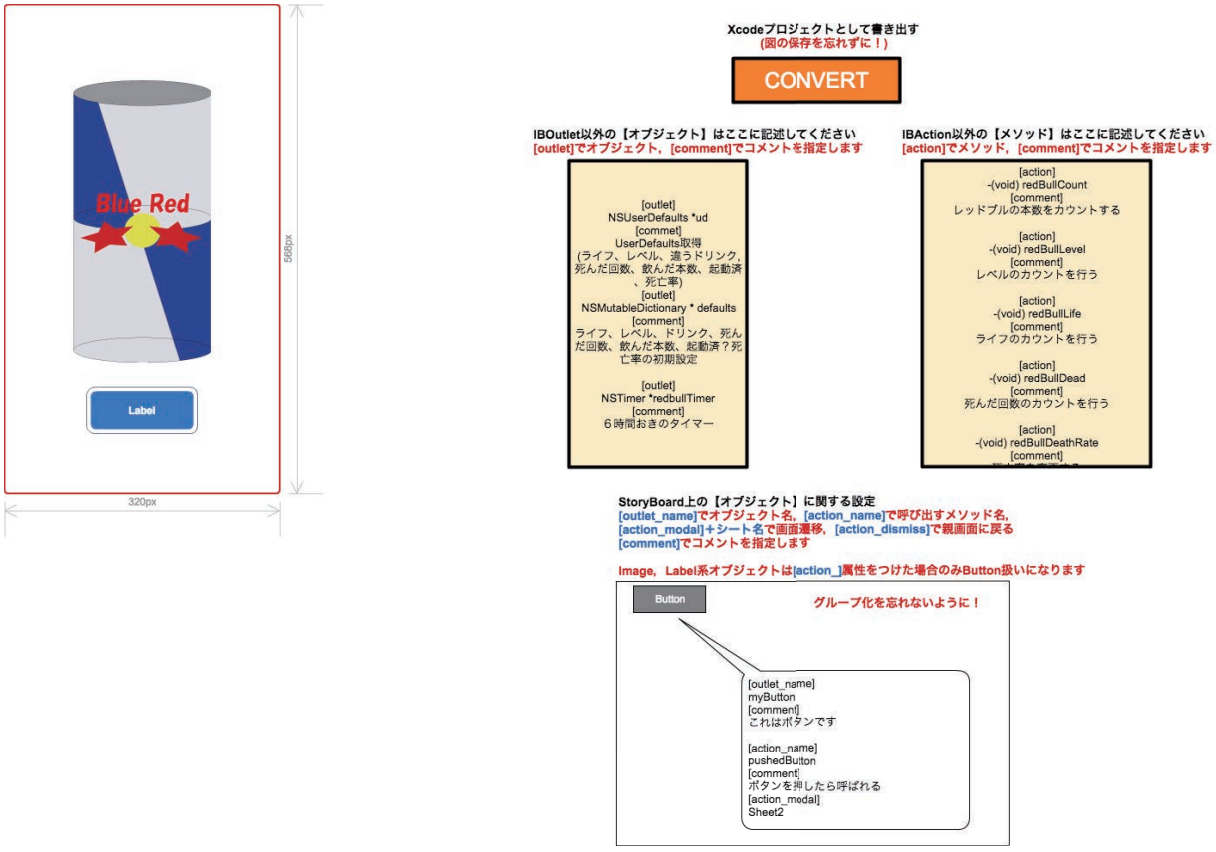

図 6.4　企画終了時にチーム A が作成した Cacoo モックアップ

図 6.5　画面共有履歴の一部（チーム A）

First View (Default pattern)　　First View (Another pattern)

図 6.6　完成したアプリケーション（チーム A）

```objc
//TODO:レッドブルの本数をカウントする
-(void) redBullCount{
    ud = [NSUserDefaults standardUserDefaults];
    NSInteger count_check = [ud integerForKey:@"KEY_COUNT"];
    count_check = count_check+1;
    _countLabel.text = [NSString stringWithFormat:@"%ld本",count_check];
    if(count_check%5==0 && dead_swich == NO){
        [self redBullLevel];
    }

    [ud setInteger:count_check forKey:@"KEY_COUNT"];
}
//TODO:レベルのカウントを行う
-(void) redBullLevel{
    ud = [NSUserDefaults standardUserDefaults];
    NSInteger level_check = [ud integerForKey:@"KEY_LEVEL"];
    level_check = level_check+1;
    [sound3 play];
    _levelLabel.text = [NSString stringWithFormat:@"Lv.%ld",level_check];
    [ud setInteger:level_check forKey:@"KEY_LEVEL"];
    [ud setInteger:3 forKey:@"KEY_LIFE"];
    _lifePic1.hidden = NO;
    _lifePic2.hidden = NO;
    _lifePic3.hidden = NO;
}
//TODO:ライフのカウントを行う
-(void) redBullLife{
    ud = [NSUserDefaults standardUserDefaults];
    NSInteger life_check = [ud integerForKey:@"KEY_LIFE"];
    if (life_check>0) {
        [sound play];
        life_check = life_check-1;
        switch (life_check) {
            case 2:
                _lifePic3.hidden = YES;
                break;
            case 1:
                _lifePic2.hidden = YES;
                break;
            case 0:
                _lifePic1.hidden = YES;
                break;

            default:
                break;
        }
        [ud setInteger:life_check forKey:@"KEY_LIFE"];
    }else{
        [self redBullDeathRate];
    }
}
```

Annotation Object

IBAction以外の【メソッド】はここに記述してください
[action]でメソッド，[comment]でコメントを指定します

[action]
-(void) redBullCount
[comment]
レッドブルの本数をカウントする

[action]
-(void) redBullLevel
[comment]
レベルのカウントを行う

[action]
-(void) redBullLife
[comment]
ライフのカウントを行う

図 6.7　完成したアプリケーションのソースコードの一部（チーム A）

6.2.3　チーム B の観察

スケッチの作成

企画段階での議論にて，チーム B が作成したスケッチを図 6.8 に示す．チーム B はこのスケッチを記述しつつ，アプリケーションのコンセプト，UI，実装方法，役割分担に関して 2 時間程度の議論を行った．チーム B の企画内容は「大学講義の内容について受講者同士がチャットできるアプリ」だった．

モックアップの作成

企画終了時にて，チーム B が作成した Cacoo のモックアップを図 6.9-6.10 に示す．チーム B は二つのモックアップ画面を作成しているが，注釈オブジェクトに関してはテンプレートシート内に一切の記述が残されておらず，またソースコード変換機能の使用履歴も確認されなかった．

開発過程での行動

チーム A に比べ，チーム B は全体的に発話が少ない印象が見受けられている．図 6.11 に画面共有履歴の一部を示す．「作業進捗共有を兼ねた画面共有機能」の「デバッグの自動検知」により，プログラマ B の画面がほぼ常に共有される状態にあったが，プログラマ A の作業画面は一度のみ共有され，プログラマ C の作業画面は一度も共有されなかったことが分かった．プログラマ A，プログラマ C らはサーバサイドの開発をメインで担当しており，開発において Xcode を活用する場面がほぼ存在しなかった．またチーム B は「共有ボタン」による画面共有を用いなかったことも判明している．開発が終盤に近づくにつれ，チーム B からは「ヤバイ」といった内容の発言が増えるようになり，その後全てのプログラマがプログラマ B の席に集まり，実装の済んでいない機能を解決しようとする動きをとった．プログラマらのこの行動により，デザイナのみ対面側の席で一人作業する状態となった．この時，デザイナはプレゼンテーション資料の作成を開始していた．その後「デバッグの自動検知」によってプログラマ B の作業画面が共有された際，画面を確認したデザイナが様子を伺いにプログラマ B の席に近づく様子が確認されている．

完成したアプリケーション

チーム B が最終的に完成させたアプリケーションの状態を図 6.12 に示す．チーム B は当初の企画においてアプリケーションの画面の数を 6 つに設定していたが，最終的に完成した

アプリケーションは 3 つの画面で構成されており，うち 1 つの画面に関しては遷移が不可能な状態となっていた．

ソースコードの状態

チーム B が最終的に完成させたアプリケーションのソースコードの一部を図 6.13 に示す．このソースコードは，アプリケーションの TOP 画面である「時間割画面」に設置されたオブジェクトに対し命名された変数名および関連付けられたメソッド名である．開発終盤において，プログラマ A は「時間割の枠が選択された際に次の画面に遷移し，時間割の選択部分によって違う内容を表示する機能」を実装しようとしていた．プログラマ A は「時間割内のすべての枠に一つずつボタンを設置し，それぞれ別のメソッドを呼び出す」ことで実装を図ろうとし，その結果このようなソースコードが記述された．その後プログラマ C が助けに入った際，このソースコードを確認したプログラマ C はこの実装方法に対し「このコードさぁ…」と苦言を述べた．チーム B は，「成果物イメージ共有を兼ねたプロトタイプ Ver.0 作成機能」のソースコード変換機能を使用していないため，ソースコードはチームの手で 1 から記述された．また，ソースコード内にチームメンバが記述したコメント文も発見されなかった．

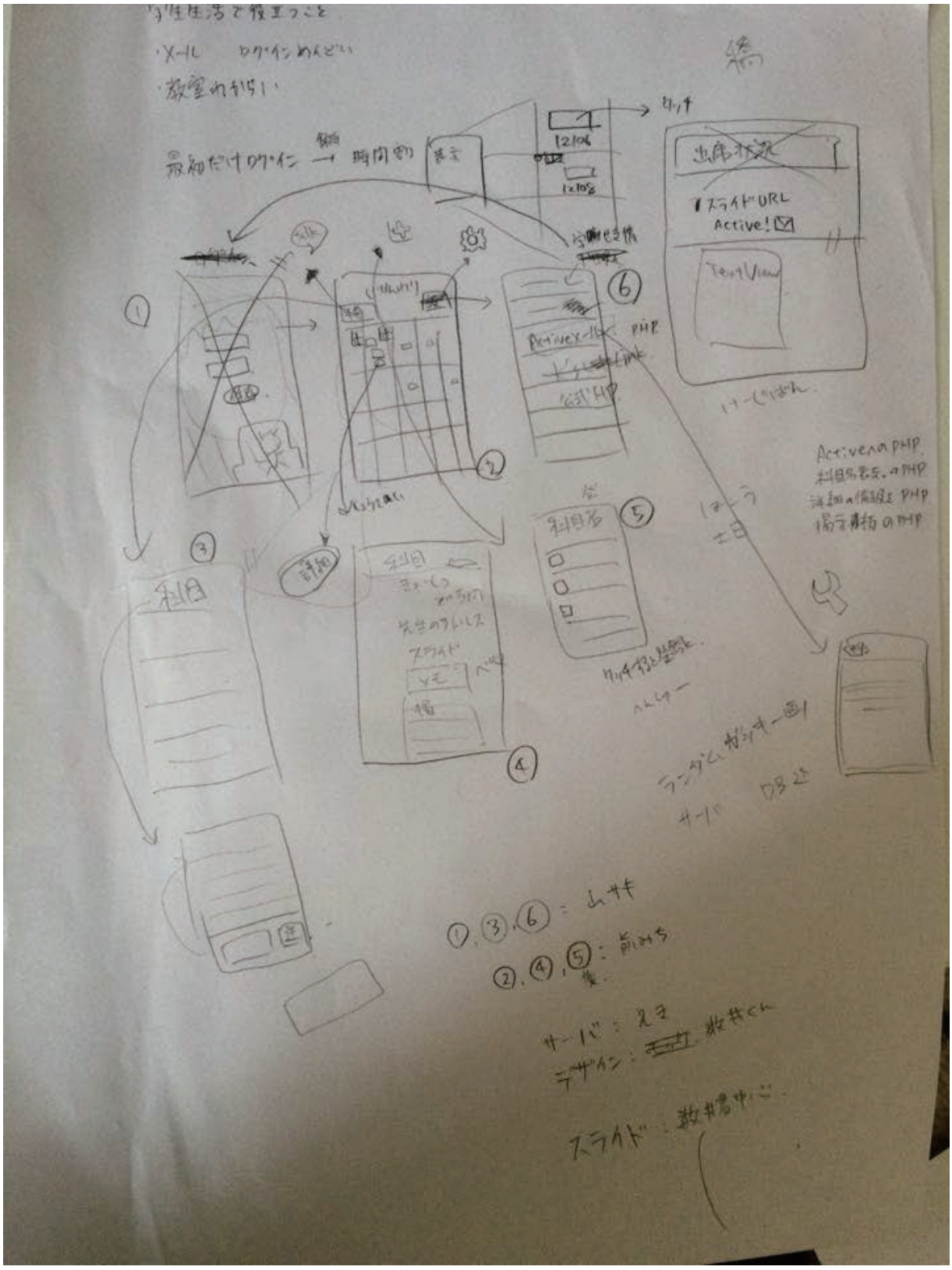

図 6.8　企画段階にチーム B が作成したスケッチ

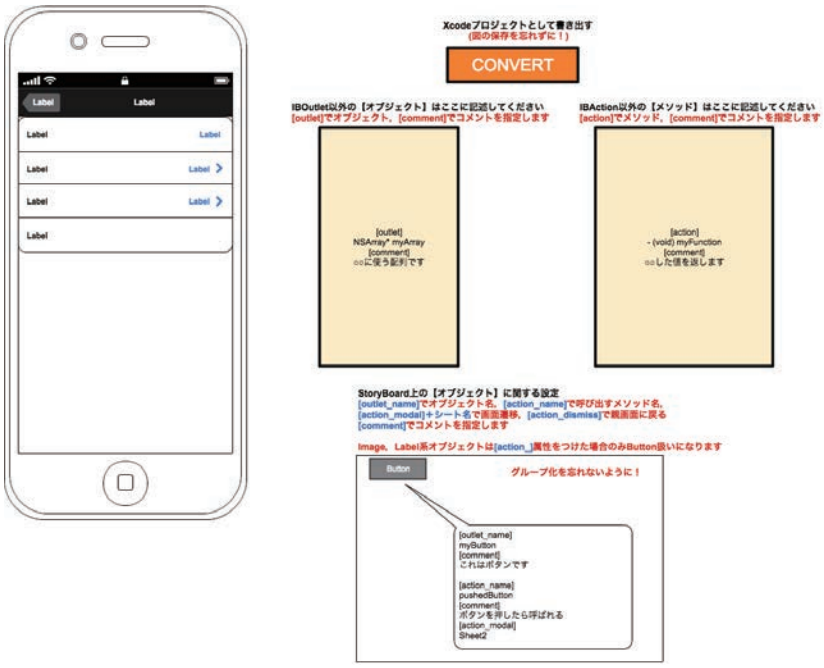

図 6.9　企画終了時にチーム B が作成した Cacoo モックアップ（1）

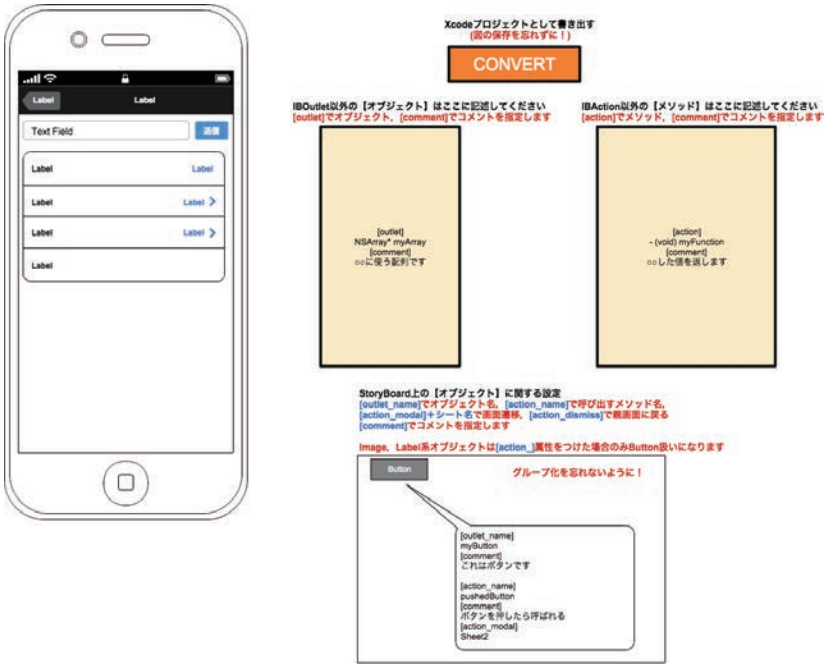

図 6.10　企画終了時にチーム B が作成した Cacoo モックアップ（2）

図 6.11　画面共有履歴の一部（チーム B）

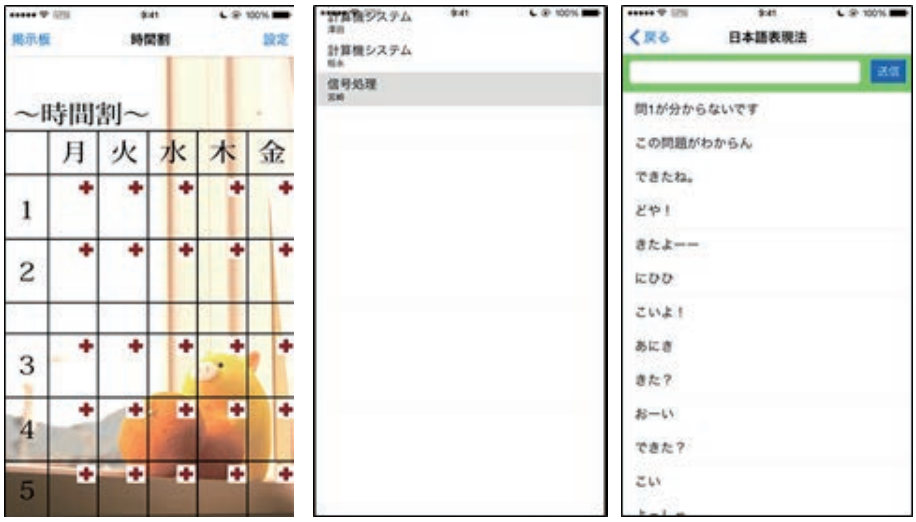

図 6.12　完成したアプリケーション（チーム B）

```
@property (weak, nonatomic) IBOutlet UILabel *label11;
@property (weak, nonatomic) IBOutlet UILabel *label21;
@property (weak, nonatomic) IBOutlet UILabel *label31;
@property (weak, nonatomic) IBOutlet UILabel *label41;
@property (weak, nonatomic) IBOutlet UILabel *label51;
@property (weak, nonatomic) IBOutlet UILabel *label12;
@property (weak, nonatomic) IBOutlet UILabel *label22;
@property (weak, nonatomic) IBOutlet UILabel *label32;
@property (weak, nonatomic) IBOutlet UILabel *label42;
@property (weak, nonatomic) IBOutlet UILabel *label52;
@property (weak, nonatomic) IBOutlet UILabel *label13;
@property (weak, nonatomic) IBOutlet UILabel *label23;
@property (weak, nonatomic) IBOutlet UILabel *label33;
@property (weak, nonatomic) IBOutlet UILabel *label43;
@property (weak, nonatomic) IBOutlet UILabel *label53;
@property (weak, nonatomic) IBOutlet UILabel *label14;
@property (weak, nonatomic) IBOutlet UILabel *label24;
@property (weak, nonatomic) IBOutlet UILabel *label34;
@property (weak, nonatomic) IBOutlet UILabel *label44;
@property (weak, nonatomic) IBOutlet UILabel *label54;
@property (weak, nonatomic) IBOutlet UILabel *label15;
@property (weak, nonatomic) IBOutlet UILabel *label25;
@property (weak, nonatomic) IBOutlet UILabel *label35;
@property (weak, nonatomic) IBOutlet UILabel *label45;
@property (weak, nonatomic) IBOutlet UILabel *label55;

- (IBAction)btn11:(id)sender;
- (IBAction)btn21:(id)sender;
- (IBAction)btn31:(id)sender;
- (IBAction)btn41:(id)sender;
- (IBAction)btn51:(id)sender;
- (IBAction)btn12:(id)sender;
- (IBAction)btn22:(id)sender;
- (IBAction)btn32:(id)sender;
- (IBAction)btn42:(id)sender;
- (IBAction)btn52:(id)sender;
- (IBAction)btn13:(id)sender;
- (IBAction)btn23:(id)sender;
- (IBAction)btn33:(id)sender;
- (IBAction)btn43:(id)sender;
- (IBAction)btn53:(id)sender;
- (IBAction)btn14:(id)sender;
- (IBAction)btn24:(id)sender;
- (IBAction)btn34:(id)sender;
- (IBAction)btn44:(id)sender;
- (IBAction)btn54:(id)sender;
- (IBAction)btn15:(id)sender;
- (IBAction)btn25:(id)sender;
- (IBAction)btn35:(id)sender;
- (IBAction)btn45:(id)sender;
- (IBAction)btn55:(id)sender;
```

図 6.13　完成したアプリケーションのソースコードの一部（チーム B）

6.2.4　アンケート

ハッカソン終了後に各参加者に対し実施したアンケート結果を述べる．実施したアンケートの内容を以下に示す．

- チームの内省評価
  - チームのマッチ度はどうでしたか？
  - チームの制作物についてどう思いますか？
  - チームのプレゼンテーションについてどう思いますか？
  - チーム内での情報共有がうまくいかなかったり，認識が食い違った場面はありましたか？
  - （あった場合）それはどのような場面でしたか？
  - 自分の作成したプログラム・素材等は最終的な制作物に反映されましたか？
  - （反映されなかった場合）それはどのようなものでしたか？
  - チーム内での大きな失敗や問題があればお聞かせください
- 「成果物イメージ共有を兼ねたプロトタイプ Ver.0 作成機能」の評価
  - 機能を活用しましたか？
  - 機能の使い勝手は良かったですか？
  - 機能は有用に感じましたか？
  - 機能が役立った場面をお聞かせください
  - 機能が役立たなかった場面をお聞かせください
- 「作業進捗共有を兼ねた画面共有機能」の評価
  - 機能を活用しましたか？
  - 機能の使い勝手は良かったですか？
  - 機能は有用に感じましたか？
  - 機能の共有レベル（0〜2）について，それぞれどのような場面で使用しましたか？
  - 機能（共有履歴表示機能を含む）が役立った場面をお聞かせください
  - 機能（共有履歴表示機能を含む）が役立たなかった場面をお聞かせください
  - システム「HackathonMediator」について意見があればお聞かせください

チームの内省評価に関するアンケート結果

■チームのマッチ度 「チームのマッチ度はどうでしたか？」という質問に対し，各チームは 5 段階（大変良かった，良かった，普通，良くなかった，大変良くなかった）で回答した．

- チーム A
  - 大変良かった：3 名
    （プログラマ A，デザイナ，プログラマ B）
  - 良かった：1 名
    （プログラマ C）
- チーム B
  - 大変良かった：1 名
    （プログラマ C）
  - 良かった：3 名
    （プログラマ A，プログラマ B，デザイナ）

■チームの制作物の内省評価 「チームの制作物についてどう思いますか？」という質問に対し，各チームは 5 段階（大変良かった，良かった，普通，良くなかった，大変良くなかった）で回答した．

- チーム A
  - 良かった：4 名
- チーム B
  - 大変良かった：1 名
    （デザイナ）
  - 良かった：2 名
    （プログラマ B，プログラマ C）
  - 良くなかった：1 名
    （プログラマ A）

■チームのプレゼンテーションの内省評価 「チームのプレゼンテーションについてどう思いますか？」という質問に対し，各チームは 5 段階（大変良かった，良かった，普通，良くなかった，大変良くなかった）で回答した．

- チーム A
  - 大変良かった：1 名
    （デザイナ）
  - 良かった：3 名
    （プログラマ A，プログラマ B，プログラマ C）
- チーム B
  - 良かった：2 名
    （プログラマ B，プログラマ C）
  - 普通：2 名
    （プログラマ A，デザイナ）

■情報共有不足の有無 「チーム内での情報共有がうまくいかなかったり，認識が食い違った場面はありましたか？」という質問に対し，各チームは 3 段階（まったく無かった，少しあった，かなりあった）で回答した．

- チーム A
  - まったく無かった：2 名
    （デザイナ，プログラマ C）
  - 少しあった：2 名
    （プログラマ A，プログラマ B）

  「少しあった」と答えた 2 名のメンバは，「（あった場合）それはどのような場面でしたか？」という質問に対し，以下の理由を述べた．
  - プログラマ A
    「詳しい仕様（どのボタンを押した時にどのような値をサーバに送るか）は口だと伝わりにくいと感じる場面があった」
  - プログラマ B
    「プレゼンテーション資料を作る際にアプリケーションの仕様がわからない時があった」
- チーム B
  - まったく無かった：3 名
    （プログラマ A，プログラマ B，プログラマ C）
  - 少しあった：1 名
    （デザイナ）

「少しあった」と答えた 1 名のメンバ（デザイナ）は，「（あった場合）それはどのような場面でしたか？」という質問に対し，「自分たちの作ったものを統合する際などにどこまで終わっているかなどを把握しきれていなかった」と述べた．

■リソースの未使用の有無 「自分の作成したプログラム・素材等は最終的な制作物に反映されましたか？」という質問に対し，各チームは 3 段階（まったく反映されなかった，反映されなかったものもある，まったく反映されなかった）で回答した．

- チーム A
  - 十分反映された：3 名
    （デザイナ，プログラマ B，プログラマ C）
  - 反映されなかったものもある：1 名
    （プログラマ A）

  「反映されなかったものもある」と答えた 1 名のメンバ（プログラマ A）は，「（反映されなかった場合）それはどのようなものでしたか？」という質問に対し，「時間が足りなくなり，予定していた画面遷移を実装することができず，用意した画像を使う機会がなかった」と述べた．
- チーム B
  - 十分反映された：2 名
    （プログラマ C，デザイナ）
  - 反映されなかったものもある：2 名
    （プログラマ A，プログラマ B）

  「反映されなかったものもある」と答えた 2 名のメンバは，「（反映されなかった場合）それはどのようなものでしたか？」という質問に対し，以下の理由を述べた．
  - プログラマ A
    「表示機能が完成せず没になった」
  - プログラマ B
    「掲示板を作ったが，「投稿後に自動更新するプログラム」を統合できなかった」

■チーム内での大きな失敗や問題 「チーム内での大きな失敗や問題があればお聞かせください」という質問に対し，各チームは以下を述べた．

- チーム A
  - プログラマ A

「『学生生活に役立つ』というテーマに対して企画のピントがずれた部分はあったと思う．最初の話し合いで企画がテーマにあっているのかを精査する必要があったと思う」

– デザイナ

「プログラマの負担が大きくなりすぎた」

- チーム B
  – プログラマ A

  「もう少し情報共有をするべきだった」
  – プログラマ B

  「最初の企画で時間をかけ過ぎてしまった．機能を盛り込み過ぎて結局削った部分が多かった」
  – プログラマ C

  「全員はじめてのハッカソンということもあって，企画に時間をかけすぎたと思う．時間をもう少しうまく使えたらよかったと思う」
  – デザイナ

  「計画の段階で, この時間でどこまでできるのかが分からず, 無理をしてしまった」

「成果物イメージ共有を兼ねたプロトタイプ Ver.0 作成機能」に関するアンケート結果

■機能の活用度 「機能を活用しましたか？」という質問に対し，各チームは 3 段階（かなり活用した，活用した，活用していない）で回答した．

- チーム A
  – 活用した：4 名
- チーム B
  – 活用した：3 名

  （プログラマ A，プログラマ C，デザイナ）
  – 活用していない：1 名

  （プログラマ B）

■機能の使い勝手 「機能の使い勝手は良かったですか？」という質問に対し，各チームは 2 段階（良かった，良くなかった）で回答した．

- チーム A

  - 良かった：4 名
- チーム B
  - 良かった：3 名
    （プログラマ A，プログラマ C，デザイナ）
  - 良くなかった：1 名
    （プログラマ B）

■機能の有用性 「機能は有用に感じましたか？」という質問に対し，各チームは 2 段階（有用だと感じられた，有用だと感じなかった）で回答した．

- チーム A
  - 有用だと感じた：4 名

「機能の役立った場面」として，メンバからそれぞれ以下のようなことが述べられている．

  - プログラマ A
    「Xcode では Storyboard にコメントをつけることができないが，この機能を使えば UI がどのような動作を担おうとしているのか分かりやすい」
  - デザイナ
    「リアルタイムで他の人と一緒に作業できた」
  - プログラマ B
    「最初にチームでソースコードを共有できたので，必要な部分だけ開発を行えた」
  - プログラマ C
    「最初に必要なメソッドなどを書き込んでおくことで，あとの開発での管理が行いやすかった」

「機能の役立たなかった場面」として，1 名のメンバ（プログラマ B）から「使用できるパーツの種類が限られていたところ」という意見が述べられた．

- チーム B
  - 有用だと感じた：3 名
    （プログラマ A，プログラマ C，デザイナ）
  - 有用だと感じなかった：1 名
    （プログラマ B）

「機能の役立った場面」として，2 名のメンバからは以下のようなことが述べられている．

– プログラマ C
「考えてるインタフェースなどを共有できた」
– デザイナ
「作った UI をすぐ見せることが出来たとき」

「機能の役立たなかった場面」として，2 名のメンバからは以下のようなことが述べられている．

– プログラマ C
「短時間だったので使い慣れなかった」
– デザイナ
「画像を張り付けるとき，要領が大きすぎてアップ出来なかったので，最大 1 メガくらいに対応して欲しい」

「作業進捗共有を兼ねた画面共有機能」に関するアンケート結果

■機能の活用度 「機能を活用しましたか？」という質問に対し，各チームは 3 段階（かなり活用した，活用した，活用していない）で回答した．

- チーム A
  – かなり活用した：1 名
  （プログラマ B）
  – 活用した：3 名
  （プログラマ A，デザイナ，プログラマ C）
- チーム B
  – 活用した：4 名

なお，プログラマ C からは「後半は一箇所の席に集まって作業することが増えたのであまり使用していなかった」という事が補足されている

■機能の使い勝手 「機能の使い勝手は良かったですか？」という質問に対し，各チームは 2 段階（良かった，良くなかった）で回答した．

- チーム A
  – 良かった：4 名
- チーム B

– 良かった：2 名
（プログラマ A，デザイナ）
– 良くなかった：2 名
（プログラマ B，プログラマ C）

■機能の有用性 「機能は有用に感じましたか？」という質問に対し，各チームは 2 段階（有用だと感じられた，有用だと感じなかった）で回答した．

- チーム A
  – 有用だと感じた：4 名

「機能の役立った場面」として，メンバからそれぞれ以下のようなことが述べられている．

  – プログラマ A
  「画像を大きく表示させることができるので意見を募りやすかった」
  – デザイナ
  「みんなで一人の画面を見に行くのは邪魔になるので，この機能は有効だと思う」
  – プログラマ B
  「プレゼンテーション資料を確認してもらう際に役立った」
  – プログラマ C
  「音楽などを聞いて作業している人に情報を伝えやすかった」

「機能の役立たなかった場面」として，2 名のメンバから以下のようなことが述べられている．

  – プログラマ A
  「プログラムコードは小さくて見えにくいので，コードに対する意見は直接プログラマの元へ向かったほうが意見交換しやすかった」
  – プログラマ B
  「通知メッセージが発行されたとき，メンバーの邪魔になる時があった」

- チーム B
  – 有用だと感じた：3 名
  （プログラマ A，プログラマ C，デザイナ）
  – 有用だと感じなかった：1 名
  （プログラマ B）

「機能の役立った場面」として，メンバからそれぞれ以下のようなことが述べられて

いる．

- プログラマ A
  「他の人にも見て欲しいときに移動せずにすぐに見せることができるので役に立った」
- プログラマ B
  「他のメンバーに大きな画面で今どこまでアプリが完成しているのかなどすぐに伝えやすい．デバッグするとき画面が勝手に反映されるのは嫌だったが，他のメンバーがすぐ見れる分には良かった」
- プログラマ C
  「作業中に自分の進捗状況を伝えるのにはかなり役立ったと思う」
- デザイナ
  「メンバーがどんなことをしてるのかがその場でわかったとき」

「機能の役立たなかった場面」として，2 名のメンバから以下のようなことが述べられている．

- プログラマ B
  「デバッグのとき自動でミラーリングされるのは恥ずかしいから選択式にしてほしい」
- プログラマ C
  「作業が落ち着いてきてからは，ミラーリングよりそのまま聞いた方が早いと思う時があった」

■共有レベル（0〜2）の活用場面 「機能の共有レベル（0〜2）について，それぞれどのような場面で使用しましたか？」という質問に対し，各チームは以下を述べた．

- チーム A
  - プログラマ A
    レベル 0：進捗や使用する画像素材を確認するとき
    レベル 1：仕様などに関する意見交換のとき
    レベル 2：大きな変更があるとき
  - デザイナ
    レベル 0：デバッグ時の自動共有のとき．結構使く便利だと感じた
    レベル 1：重要だと思う報告のとき．集中している人に画面を注目させる上で非常に有効だった

レベル 2：はじめに議論した部分に対する重要な変更点の報告のとき
  - プログラマ B
    レベル 0：ミラーリング機能の動作チェックのために使った
    レベル 1：重要度の低いと思われる内容を共有するとき．一番利用した
    レベル 2：絶対に確認してもらいたいとき
  - プログラマ C
    レベル 0：使わなかった
    レベル 1：デバッグするとき
    レベル 2：どうしても見てもらいたい状況になったとき
- チーム B
  - プログラマ A
    レベル 0：とりあえずできているかを自分で確認するとき
    レベル 1：こんな感じになっていると情報を共有するとき
    レベル 2：必ず情報を共有したいとき
  - プログラマ B
    レベル 0 しか使用しなかった，デバッグするときに自動で共有された
  - プログラマ C
    レベル 0：使わなかった
    レベル 1：デバッグするとき
    レベル 2：うまく動いたことを報告するとき
  - デザイナ
    レベル 0：試作とかで作ったとき
    レベル 1：大まかに完成したとき
    レベル 2：作品として完成したとき

「HackathonMediator」に対する意見および感想

「システム「HackathonMediator」について意見があればお聞かせください」という質問に対し，各チームは以下を述べた．

- チーム A
  - プログラマ A
    「ハッカソンという活動に初めて参加し重要だと感じたのは，お互いの意思疎通だった．その点でこのシステムはコミュニケーションを取る機会を増やしてくれ

るため有用に感じた．ハッカソンのみならず他のイベントにも応用できるシステムではないかと思う」

  - デザイナ

    「他者とコミュニケーションを図る上でとても有効だと感じた」

  - プログラマ B

    「作業中にいきなり青いディスプレイになるのでビックリする．画面共有する前に予告してくれる機能が欲しい」

  - プログラマ C

    「より実用性があがって，各ハッカソンイベントで利用されるようなツールになることを期待する」

- チーム B
  - プログラマ A

    「デバッグの際に少しだけログを確認したいときにも画面共有してしまうので，『共有しない』という選択肢が欲しかった」

  - プログラマ B

    「デバッグを停止したタイミングで共有も一緒に終了させてほしい」，「結局，情報共有はお互い作業しながら話せばいいし，画面を見に行った方が早いし，プロジェクトも直に作る方が断然早いと思う」

  - プログラマ C

    「使い方を熟知した上でなら使いやすいんだろうなと思う．時間に追われた作業だったので活用する余裕が無かった」

  - デザイナ

    「人数がとても多い場合や，少し離れて作業している場合などにはかなり有効だと思った」

第 7 章

考察

この章では，第 5 章で述べた第一次実験の結果から「作業進捗共有を兼ねた画面共有機能」のプロトタイプの効果に関する考察を述べたのちに，第二次実験の結果から「成果物イメージ共有を兼ねたプロトタイプ Ver.0 作成機能」と「作業進捗共有を兼ねた画面共有機能」に関する考察をそれぞれ述べ，最後にシステム「HackathonMediator」の展望を述べる．

7.1　第一次実験での考察

「作業進捗共有を兼ねた画面共有機能」のプロトタイプを導入したことによる考察を述べる．チーム B，チーム C のそれぞれの観察から，チーム内の発話の少なかったメンバに対し本機能がコミュニケーションの機会をもたらしたことを確認した．また，機能の効用に関するアンケートなどを通して，画面の共有によって「ディスカッションが増えた」とする意見が多く得られており，本機能によってチーム内でのコミュニケーションを生み出す作用をもたらすことができたと考えられた．これらのことから，本機能は「ソロプレイヤー化」をはじめとして「他メンバへの関心の希薄化」の問題解決に貢献することができたと考えられる．このほか，「メンバーに注目して欲しいタイミングで注目させられるのが良かった」，「デザイナの絵をみんなで見るとき便利だった」といった肯定的な意見を得られた．

「機能の使い勝手」に関しては，11 名中 10 名が「便利だった」，「現状の状態を理解しやすかった」など肯定的な反応を示しているが，チーム B のデザイナのみが「あまり有用に感じられなかった」と述べている．この理由として，デザイナは効果音などの音響編集を主な作業としていたことが挙げられる．更にこのデザイナは音響編集を行う上で作業 PC とは異なる専用機器を活用していたため，「リソース作成・編集の自動検知」は作用しなかった．

また，アンケート調査において，「カクつきが気になった」という回答が複数見受けられ

ている．これには，30 時間連続して行われた開発作業と，開発環境として使用された Unity の問題が大きく影響していると考えられる．主に 3D ゲーム開発で用いられる Unity は，開発ツールの中でも高度な演算処理を必要とするため，長時間に渡る活用によって各メンバの作業 PC には大きな負荷が発生していた．特にチーム A とチーム B は動きの激しいゲームを開発していたため，この「カクつき」によってゲームの動きが伝わりにくくなる問題が次第に見受けられるようになった．開発終盤においては，チーム A のメンバがデバッグ画面を見せる際，大型ディスプレイではなく自身のディスプレイを直接見せるようになっていた．

7.1.1　機能の改善

■画面共有時の効果音の変更　本機能に対し，「悪いと感じた点」として「音が耳障り」だという指摘が多かった．これは画面共有を行ったことをメンバに伝えるための要素として設定した効果音であったが，ハッカソンの最中に「音が大きい」，「一回あたりの音が長い」などの苦言を発する参加者も多かった．第二次実験の際，この効果音は音量および音の長さを小さくしたものに変更した．

■ひとことコメントの実装　ハッカソンの最中において，チーム B のデザイナは「表示された画面を見ても，何をやってるのかが分かりにくい」，「作業の文脈が分からない」と述べた．第 5 章で紹介した「ひとことコメント」や「画面共有履歴」はこの指摘を受けて開発している．

■画面共有の停止機能の実装　ハッカソンの最中において，チーム C のプログラマ B は「画面が表示されっぱなしになるのはいやだ」と述べた．第 5 章で紹介した「共有停止ボタン」はこの指摘を受けて開発している．

■秒読みダイアログの実装　「デバッグの自動検知」や「リソース作成・編集の自動検知」によって画面共有が行われた際，共有者本人がそのことに気付かなかった事態が複数のチームで確認された．第 5 章で紹介した「秒読みダイアログ」はこの問題を受けて開発しており，画面共有の直前に「あなたの画面が〇〇秒後に共有されます」といった内容を通知する．

7.2　第二次実験での考察

HackathonMediator の二つの機能について，第二次実験の結果から考察を述べる．

7.2.1　「成果物イメージ共有を兼ねたプロトタイプ Ver.0 作成機能」に関する考察

第二次実験において，チーム B はあくまでも Cacoo の提供している機能を活用したのみに留まっており，ソースコード変換機能を使用しなかったため，本機能を「活用した」といえるのはチーム A のみであった．

7.2.2　チーム A からの考察

チーム A の行動観察から，スケッチの作成段階で議論されていた機能が，最終的に完成したアプリケーションの中に反映されていることを確認した．このアプリケーションの「根幹部分」は，プロトタイプ Ver.0 内で定義された UI やメソッドの内容に基づいて実装されていたことがソースコードから確認できている．この「根幹部分」をベースとして，アプリケーションには「パラメータをサーバに保存する機能」や「効果音を再生する機能」などの機能拡張が施されていた．また，アンケートからは「最初にチームでソースコードを共有できたので，必要な部分だけ開発を行えた」,「最初に必要なメソッドなどを書き込んでおくことで，あとの開発での管理が行いやすかった」といった開発時に得られたメリットも述べられており，このようなことから，プロトタイプ Ver.0 の作成過程での作業が，開発時における「優先順位の誤り」などを防止する役目を果たしたのではないかと考察する．

本機能の有用性に関するアンケートでは，チーム A の全員が「有用だと感じた」と述べており,「この機能を使えば UI がどのような動作を担おうとしているのか分かりやすい」,「リアルタイムで他の人と一緒に作業できた」などの感想から，本機能が「成果物イメージの差異」や「他メンバへの関心の希薄化」の問題に対し解消をもたらす可能性が見受けられた．

7.2.3　チーム B からの考察

チーム B に関しては，主に「本機能が活用されなかった理由」について考察する．チーム A の企画段階では，主に「コンセプト，プレゼンテーション，機能」についてのみが議論されたことに対し，チーム B は企画段階でのスケッチにて「コンセプト，機能，画面のデザイ

ン，実装方法，役割分担」に至るまでが一通り議論されていた．なお，機能や実装方法などに関する議論が長引き，チーム B の議論はチーム A に比べて 1 時間程度長引いている．その後チーム B は，Cacoo でのモックアップ作成段階にて 2 つの画面を作成しチーム内で共有を図っていたが，この中にアプリケーションの TOP 画面となった「時間割画面」が存在していないなど，作られたモックアップは完全なものではなかった．アンケートにて「企画に時間をかけすぎた」ことや，「機能を盛り込みすぎた」ことがメンバから述べられ，また「時間に追われた作業だったので活用する余裕が無かった」といった回答から，チーム B においては「開発への焦り」に関連する問題が発生したと思われた．本来であれば，企画段階での「画面のデザイン，実装方法，役割分担」に関する議論は，本機能を活用する中で行われるべきであったと考えられ，チーム B がコンセプトや機能について議論した際に，「続きは本機能を用いて議論すること」をアナウンスすべきだった．

以上の内容を踏まえた上で，第 3 章で述べた Jørgensen ら [22] の「プロトタイピングツールに望まれる要素」に基づいて本機能を評価した．

1. 迅速かつ簡単に作れること
   モックアップツールとして提供されている Cacoo においては，アプリケーションの UI パーツがステンシルテンプレートによって提供されているほか，共同編集が可能である．本機能が提供するソースコードの自動生成の機能を併用することにより，画面のデザインにおいては「迅速さ」を提供できていると考えられた．一方で，本機能を活用してもらう上で独自に設けた「タグを用いた注釈の記述方法」などのルールに関しては，チーム B から「使い方を熟知した上でなら使いやすいんだろうなと思う」といった意見があるように，初めて使用するユーザにとって「複雑さ」を感じさせてしまったと思われる．今後は注釈の記述方法について再考し，よりストレスのないインタフェースを実現することを目指したい．
2. 実際のシチュエーション（利用者の生活の中でのワンシーン）を想定してテストできること
   本機能では，Cacoo で作成したモックアップをソースコードに変換することを可能としており，この機能を用いて得られるプロトタイプ Ver.0 を実際のネイティブアプリケーションとしてハードウェアにインストールできる．そのため実際のシチュエーションでのテストが可能である．ゆえにこの条件はクリアされた．
3. 実際のハードウェア上で見せることが可能であること
   本機能を用いてプロトタイプ Ver.0 を実際のハードウェアにインストールすることが可能である．ゆえにこの条件はクリアされた．

4. デザインの説明にファシリテータの存在を必要としないこと
   本機能を用いてプロトタイプ Ver.0 を実際のハードウェアにインストールすることが可能である．ユーザがハードウェアさえ所持していれば，誰でもデザインを体験することができる．ゆえにこの条件はクリアされた．
5. 現実に対する高い忠実度と状態遷移を可能にすること
   本機能を用いてプロトタイプ Ver.0 を実際のハードウェアにインストールすることが可能である．Cacoo でモックアップを作成する際に「タグを用いた注釈の記述方法」を用いて状態遷移を記述しておくことで，プロトタイプ Ver.0 に状態遷移の機能が付加される．プロトタイプ Ver.0 が提供する状態遷移は現実のアプリケーションそのものである．ゆえにこの条件はクリアされた．

7.2.4 「作業進捗共有を兼ねた画面共有機能」に関する考察

第一次実験では，「デバッグの自動検知」や「リソース作成・編集の自動検知」での画面共有によって，「他メンバへの関心の希薄化」や「ソロプレイヤー化」の問題を防止する効用が見られた．第二次実験においても，共有された画面を確認したメンバが意見や感想を述べたり，様子が気になり近くに駆けつける場面が双方のチームで確認されている．第一次実験のような場面を再び確認できたことから，本機能はチームに対し情報共有の機会をもたらす作用が存在すると考えられ，「情報共有の頻度低下」や「共有された情報の把握不足」の問題を解消できる可能性が見受けられた．しかし，本機能が目指した「チーム内での作業進捗の共有を確実なものにする」ことに関しては，複数の課題点が確認されている．

課題の一つは，「デバッグの自動検知」や「リソース作成・編集の自動検知」の可能な範囲に限界が存在したことである．第二次実験では双方のチームがサーバサイドと連携する iPhone アプリケーションを開発しており，サーバサイドのプログラマの作業に関しては「デバッグの自動検知」のサポートを受けられなかった．チーム A のデザイナにおいても，リモートマシン上でイラストツールを使っていたため，「リソース作成・編集の自動検知」のサポートを受けられなかった．この問題に対し，チーム A のメンバは自発的に「共有ボタン」を用いることで画面を見せる行動をとっていたが，一方のチーム B においては「共有ボタン」による自発的な画面共有が一度も行われていなかった．

もう一つの課題として，プライバシーなどの心理面の問題も考えられる．両実験にて，「必要がないときも画面が共有される」，「画面上で見せる範囲の制御が出来ない」，「自動でミラーリングされるのは恥ずかしい」など，画面共有そのものに拒否感を抱く意見が述べられた．この問題に関しては「公開可能な画面の範囲」などを設定可能にすることを今後検討し

ていく．

また，「通知メッセージの表示が相手の邪魔になる」ことに抵抗感を抱くメンバも確認されている．本機能では，画面共有時の「通知メッセージ」が時としてメンバに不利益をもたらす可能性を考慮し，3 段階の「共有レベル」を設けていた．レベル 0 においては通知メッセージが発行されないため，作業場面の重要度などに合わせてレベルを使い分けてもらうようにしていたが，レベル 0 が必要以上に活用された可能性が高く，重要な情報の共有機会を逃してしまった可能性も考えられた．

7.3　システム「HackathonMediator」の展望

今後 HackathonMediator をハッカソン支援システムとして導入していく上で，考えられる展望について述べる．

7.3.1　活用メリットの理解度向上

HackathonMediator をハッカソンチームに導入する上では，活用のメリットを十分に理解させることが重要であると考察する．第二次実験では，全ての参加者がハッカソンについて「経験したことがない」と答えていた．アンケート結果にて，チーム A のプログラマ A は「ハッカソンという活動に初めて参加し重要だと感じたのは，お互いの意思疎通だった」と述べたほか，チーム B のプログラマ B は HackathonMediator の機能について「有用だと感じなかった」とした上で，「結局，情報共有はお互い作業しながら話せばいいし，画面を見に行った方が早いし，プロジェクトも直に作る方が断然早いと思う」と述べており，システムを利用するメリットについて理解を示さなかった．このようなことから，ハッカソンの初心者にはそもそも失敗の経験が存在しないため，「何が大事なのか，何を避けるべきなのか」について事前に十分な理解を促し，提案システムの活用メリットを訴える工夫が必要だと考えられた．

7.3.2　複数のツールなどとの併用可能性

ハッカソンの最中では，HackathonMediator のみならずその他のツールやプロセスが併用される可能性が想定され，それらの影響からチームに「不利益」が生み出される可能性にも気を配る必要があると考えられた．第一次実験の際，「作業進捗共有を兼ねた画面共有機能」に対し，Unity を開発に用いたメンバからは「カクつきが気になった」という意見が多く，これは Unity が長時間に渡って作業 PC に負荷をもたらした影響によるものと思われた．ま

た，Unity を用いない場合においても，複数のツールを長時間稼働させた状態が続けば，いずれ同様の問題が発生する可能性はある．ハッカソンでの活用を前提としたシステムを設計する上では，可能な限り「他のツールなどに悪影響を受けない・もたらさない」ことを踏まえるべきなのかもしれない．

7.3.3　様々なハッカソンチームでの評価

今回実施した評価は，非常にわずかな被験者数に対して実施した行動観察とアンケートに基づくものであった．ハッカソンチームの失敗要因をはじめ，HackathonMediator の効用を定量的に明らかにする上で，さらなる被験者の協力が必要である．目下実施すべき調査内容は，ハッカソンの開催期間やチームメンバ数など，「開催条件」が異なる場合での効用の違いを明らかすることであると考えられる．特に「成果物イメージ共有を兼ねたプロトタイプ Ver.0 作成機能」に関しては，30 時間などの長時間のハッカソンにおける評価も検討したい．ハッカソンで開発されるアプリケーションの「複雑さ」は，ハッカソンの開催期間に比例すると考えられ，「より複雑なアプリケーション」と「より長時間の開発」において本機能を活用した場合，確認できる作用はまた異なるものとなる可能性も大いにある．

今後，より多くのハッカソンチームに本システムを導入してもらうために，活用メリットの理解度向上（節 7.3.1）や，複数のツールなどとの併用可能性（節 7.3.2）などに配慮した改善を施していきたい．

第 8 章

まとめ

本研究では，ハッカソンチームにおける協調型開発の失敗要因について調査を行い，それらを解決するための支援システム「HackathonMediator」（意：ハッカソンチームの仲介者）を提案し，実際のハッカソン内で導入・評価した．ハッカソンは「多くの人々がノンストップで一つの物事に熱中して取り組む」ことを指す開発イベントであり，3-6 名程度のグループ（多くの場合が即席である）が 1-2 日ほどの短期間内で「企画・開発・プレゼンテーション」を行い，革新的な成果物を生み出すことが目標とされる．ハッカソンには非常に短期的な時間制限が設けられているため，状況に合わせた的確な意思決定やスケジューリングなどがハッカソンチームの成功に大きく起因すると考えられる．共同開発における支援手法は古くから研究されている分野であるが，従来の支援システムやプロセスのほとんどは，長期的なプロジェクトや，企業など確立された組織での活用を前提としたものが多く，企画・開発・プレゼンテーションを内包した濃厚かつ高速な作業や，即席のチーム結成が想定されるハッカソンへの導入に焦点を当てた支援手法の研究事例は少ない．また，ハッカソンおよび同様の形態を持つイベントに関する研究報告は近年少しずつ増加してきているものの，既存の研究報告はイベント運営者としてのケーススタディが多く，ハッカソンチームの内部で起きていた事象については明らかにされていない点が多い．本研究においてハッカソンチームの失敗要因を調査する上では，ハッカソンチーム内部での活動に注目する必要があった．そのため本研究では，予備実験において 4 度開催したハッカソンを基に，「開発への焦り」による「成果物イメージの差異」，「他メンバへの関心の希薄化」による「情報共有の頻度低下」や「共有された情報の把握不足」が根本的な問題であると仮定した．ハッカソンの支援システムにおいては「開発に遅延なく成果物の完成イメージを明確化・共通化する」ことと，「チーム内での作業進捗の共有を確実なものにする」ことが重要であると考えられたため，提案システム「HackathonMediator」においては，「成果物イメージ共有を兼ねたプロトタイ

プ Ver.0 作成機能」および「作業進捗共有を兼ねた画面共有機能」を提供した．第一次実験では，「作業進捗共有を兼ねた画面共有機能」のプロトタイプを導入し，「デバッグの自動検知」や「リソース作成・編集の自動検知」を用いた画面共有によって，チーム内の発話の少なかったメンバに対し本機能がコミュニケーションの機会をもたらされたことなどから，「他メンバへの関心の希薄化」や「ソロプレイヤー化」の問題を防止する効用を確認した．第二次実験では，「成果物イメージ共有を兼ねたプロトタイプ Ver.0 作成機能」により，「成果物イメージの差異」や「他メンバへの関心の希薄化」，「優先順位の誤り」などの問題を防止できる可能性が見受けられた．また，「作業進捗共有を兼ねた画面共有機能」により，共有画面を確認したメンバが意見や感想を述べたり，様子が気になり近くに駆けつけるなど，第一次実験で確認した作用を再び確認できた．このことから，本機能はチームに対し情報共有の機会をもたらしており，「情報共有の頻度低下」や「共有された情報の把握不足」の問題を解消できる可能性が見受けられた．

ハッカソンは，近年より教育分野や産業発展への導入可能性が検討されはじめた新興的な開発イベントである．本研究から得られた「ハッカソンチームの失敗要因」および「ハッカソンチームを成功に導くための解決手法」に関する知見が，今後のハッカソンの発展における一助となることに期待したい．

謝辞

本研究を修士論文としてまとめるにあたり，多岐にわたって熱心なご指導と激励をいただきました，西本一志先生（北陸先端科学技術大学院大学ライフスタイルデザイン研究センター）に心からの感謝の意を表明いたします．また，（株）リクルートホールディングス R&D 戦略室 メディアテクノロジーラボの伴野智樹氏には，ハッカソンの見学機会を含め，ハッカソンに関連する様々な情報を提供いただきました．（株）ギブリー社員の皆様には，ハッカソンの開催に関する情報提供のほか，本研究が提案するアプローチに関して多くの重要なアドバイスをいただきました．双方様のご協力により，本研究に取り組むにあたってより多くの視点を取り入れることができました．ここに心からの感謝の意を表明いたします．九州産業大学でのハッカソン開催にあたっては，田中康一郎先生（九州産業大学情報科学部教授），田中先生の研究室に所属する学生の皆様に全面的なご協力を頂きました．お忙しい中でのご厚意に多大なる感謝を申し上げます．最後に，西本一志先生の研究室に共に所属し，日頃より多くのアドバイスを頂戴した北山史朗氏，久留島寛也氏，長谷部礼氏，村井孝明氏，森本康希氏をはじめ，ハッカソンを学内で開催するにあたってご協力いただきました多くの参加者の皆様に対し，ここに感謝の意を表します．ありがとうございました．

参考文献

[1] 渡辺信一. 国のオープンデータ政策と自治体のオープンガバメントに向けた取り組み-オープンデータの活用による自治体行政の展開に向けて. *NRI* パブリックマネジメントレビュー, 131:8–16, 2014.

[2] 清水たくみ. オープンデータ活用によるアプリケーション開発. 組織学会大会論文集, 2(1):38–43, 2013.

[3] 株式会社ローソン. Hackalawson 2013. http://www.lawson.co.jp/campaign/static/hackalawson/.

[4] OpenBSD. Openbsd hackathons. http://www.openbsd.org/hackathons.html.

[5] Jorge Luis Zapico Lamela, Daniel Pargman, Hannes Ebner, and Elina Eriksson. Hacking sustainability: Broadening participation through green hackathons. In *Fourth International Symposium on End-User Development. June 10-13, 2013, IT University of Copenhagen, Denmark*, 2013.

[6] ChatWork. http://www.chatwork.com/.

[7] slack. https://slack.com/.

[8] hackpad. https://hackpad.com/.

[9] 株式会社ヤフー. The open hack day 2. http://yhacks.jp/ohd2/.

[10] 株式会社グリー. 千葉大学教育学部とグリー、「メディアリテラシー教育演習」授業を共同で実施. http://corp.gree.net/jp/ja/news/press/2013/1001-01.html.

[11] ITNAV. 石巻ハッカソン、今年も開催！！http://itnav.jp/archives/977.

[12] 東京証券取引所. 【ソーシャルかぶコン 2013】“東証初”のイベント！ 「ハッカソン」開催のお知らせ！http://www.tse.or.jp/news/48/130913_c.html.

[13] NHK. Nhk hackathon. http://www.nhk.or.jp/kokusaihoudou/hackathon/.

[14] 青森県庁. 農業×ＩＴ マッチングワークショップを開催します . http://www.pref.aomori.lg.jp/sangyo/energy/IT_WS_02agri.html.

[15] COSPLAYTHON. Cosplaython2014. http://worldcosplaysummit.jp/

appcontest/cosplaython.html.

[16] M Salman Bashir and M Rizwan Jameel Qureshi. Hybrid software development approach for small to medium scale projects: Rup, xp & scrum. *Cell*, 966:536474921, 2012.

[17] Aaron Shaw, Haoqi Zhang, Andrés Monroy-Hernández, Sean Munson, Benjamin Mako Hill, Elizabeth Gerber, Peter Kinnaird, and Patrick Minder. Computer supported collective action. *interactions*, 21(2):74–77, 2014.

[18] Sharon Ryan and Rory V O' Connor. Acquiring and sharing tacit knowledge in software development teams: An empirical study. *Information and Software Technology*, 55(9):1614–1624, 2013.

[19] Irwin Kwan, Daniela Damian, and M Storey. Visualizing a requirements-centred social network to maintain awareness within development teams. In *Requirements Engineering Visualization, 2006. REV'06. First International Workshop on*, pages 7–7. IEEE, 2006.

[20] Mikko Raatikainen, Marko Komssi, Vittorio Dal Bianco, Klas Kindstom, and Janne Jarvinen. Industrial experiences of organizing a hackathon to assess a device-centric cloud ecosystem. In *Computer Software and Applications Conference (COMPSAC), 2013 IEEE 37th Annual*, pages 790–799. IEEE, 2013.

[21] Juergen Musil, Angelika Schweda, Dietmar Winkler, and Stefan Biffl. Synthesized essence: what game jams teach about prototyping of new software products. In *Software Engineering, 2010 ACM/IEEE 32nd International Conference on*, volume 2, pages 183–186. IEEE, 2010.

[22] Anders P Jørgensen, Matthijs Collard, and Christian Koch. Prototyping iphone apps: realistic experiences on the device. In *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries*, pages 687–690. ACM, 2010.

[23] José Matías Rivero, Gustavo Rossi, Julián Grigera, Esteban Robles Luna, and Antonio Navarro. From interface mockups to web application models. In *Web Information System Engineering–WISE 2011*, pages 257–264. Springer, 2011.

[24] Bill Buxton. Sketching user experiences: getting the design right and the right design (interactive technologies). *Burlington, MA: Morgan Kaufmann*, 2007.

[25] Mark Zarb, Janet Hughes, and John Richards. Industry-inspired guidelines improve students' pair programming communication. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*, pages 135–140. ACM, 2013.

[26] NodeRabbit Inc. 10 tips for hackathon success. http://appsembler.com/blog/10-tips-for-hackathon-success/.

[27] Cacoo. https://cacoo.com/.

[28] join.me. https://join.me/.