CEDEC2010
CESA Developers Conference

リードプログラマへの誘い

株式会社アルファ・システム
開発部開発課
深澤　正俊

自己紹介

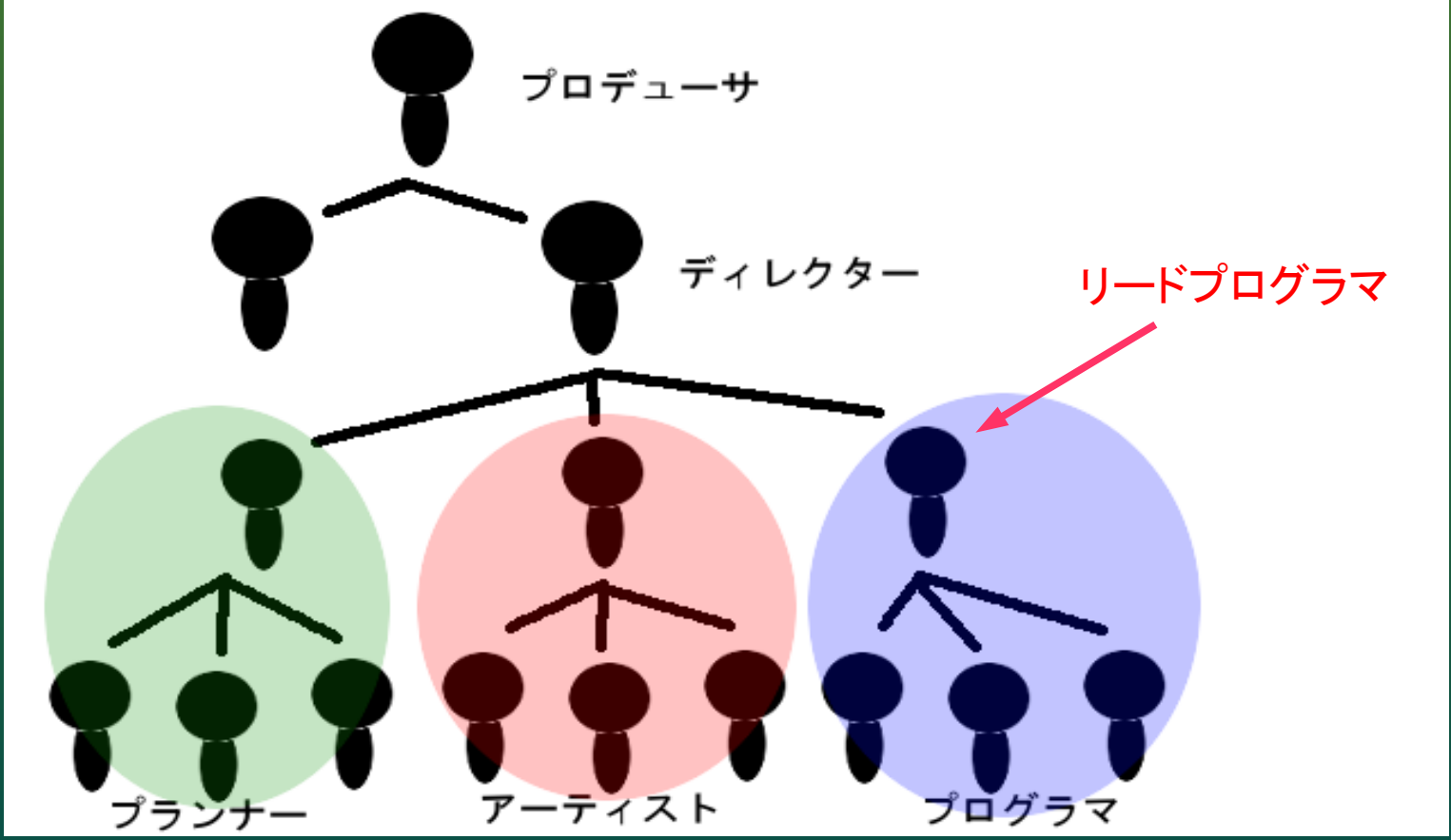
- 深澤　正俊（ふかざわ・まさとし）
  - 株式会社アルファ・システム勤務
    - 所在地は熊本
    - 「俺の屍を超えてゆけ」「高機動幻想ガンパレードマーチ」
    - 最近では「ファンタシースターポータブル」「TOW」等
  - 経歴
    - コンシューマー向けゲーム開発にプログラマとして従事。
    - １１年目（満１０年５ヶ月位）
    - ゲームフレームワーク部分、コンテンツ開発環境構築、開発プロセスマネージメントが専門領域。
    - 純粋なリードプログラマ経験は3案件、補助案件や実質的リードプログラマは入社3年目くらいから担当。

Agenda

- リードプログラマとは何か？
- リードプログラマの実務例
- 質疑応答

「リードプログラマ」の存在理由

- まずは「リードプログラマ」が存在している理由を考察してみましょう。



一人の力では立ち行かない

- 「実作業時間＜必要作業時間＜締切までの時間」
  - 人の一生は大規模なゲームを作るのには足りない。
    - 人生 80 年としたら、フルに使って一人でやったとしても、50 人規模、1 年半程度の仕事。
  - 1 日の稼動時間は 24 時間を超えられない。
    - 締め切りとの兼ね合いで 1 日 24 時間以上の作業が必要になっている時点でゲームオーバー。

「実作業時間＞必要作業時間」の右辺

- 必要作業時間を減らす
  - 効率化
    - ターンアラウンドタイムの縮減　(例)ビルド時間の短縮
    - 自動化　(例) ソースコード追加にあたり Wizard の作成
  - 開発規模の低減
    - ゲームデザインのパターン化
      - （例）シリーズ化
      - 飽きられやすくなるという欠点も。
    - 仕様の削減
      - やりすぎるとゲームとして成立しなくなる。

「実作業時間＞必要作業時間」の左辺

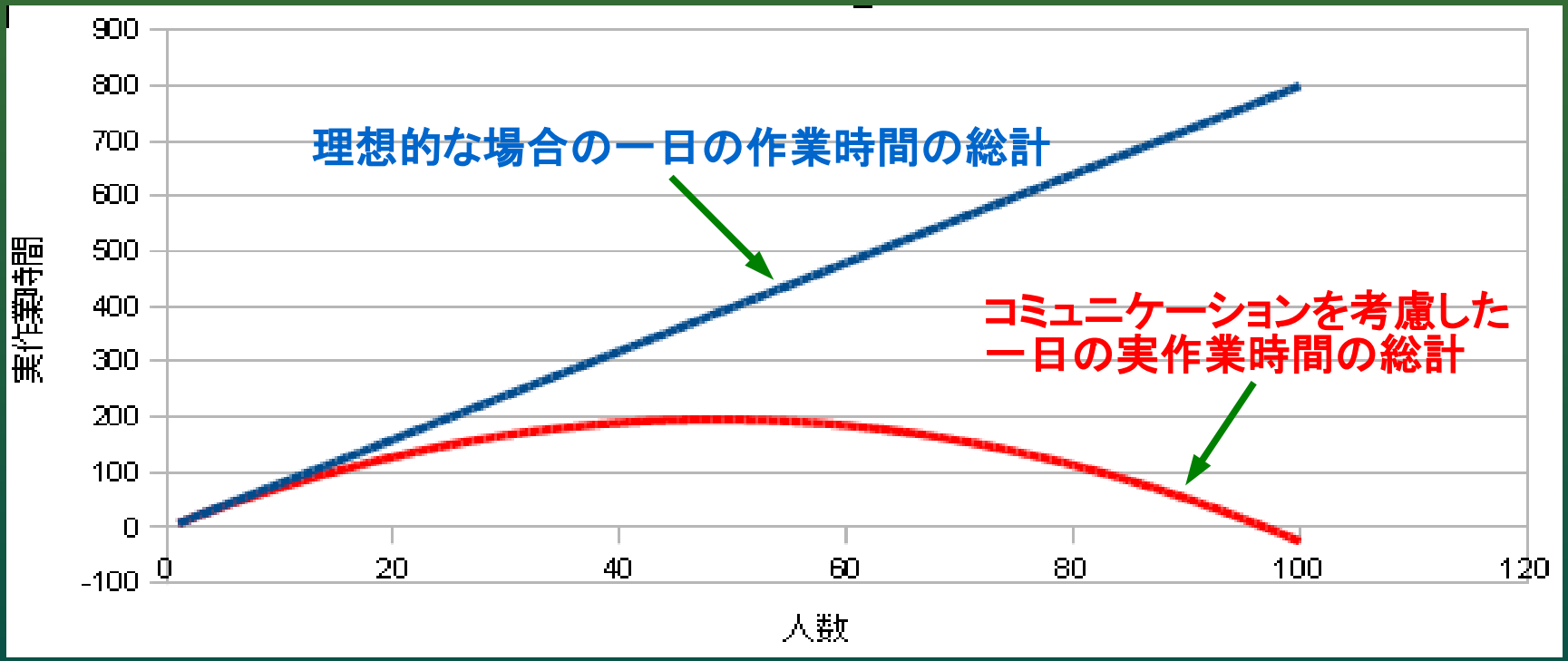
- 実現できる仕事量を増やす
  - 簡単に言えば「作業する人を増やしましょう」

●プロジェクトの人員が増えると…

- 実作業時間は単純に比例して増加(→O(N))
- 目的を共有するためにはコミュニケーションが必要
  - 何も管理されていない状況では全員が全員とコミュニケーションを取る必要がある。
    - (例) 仕様変更の告知、ツール更新の告知…
  - n(n-1)/2 組の相互コミュニケーションが発生する。
    - O(N^2) のオーダー

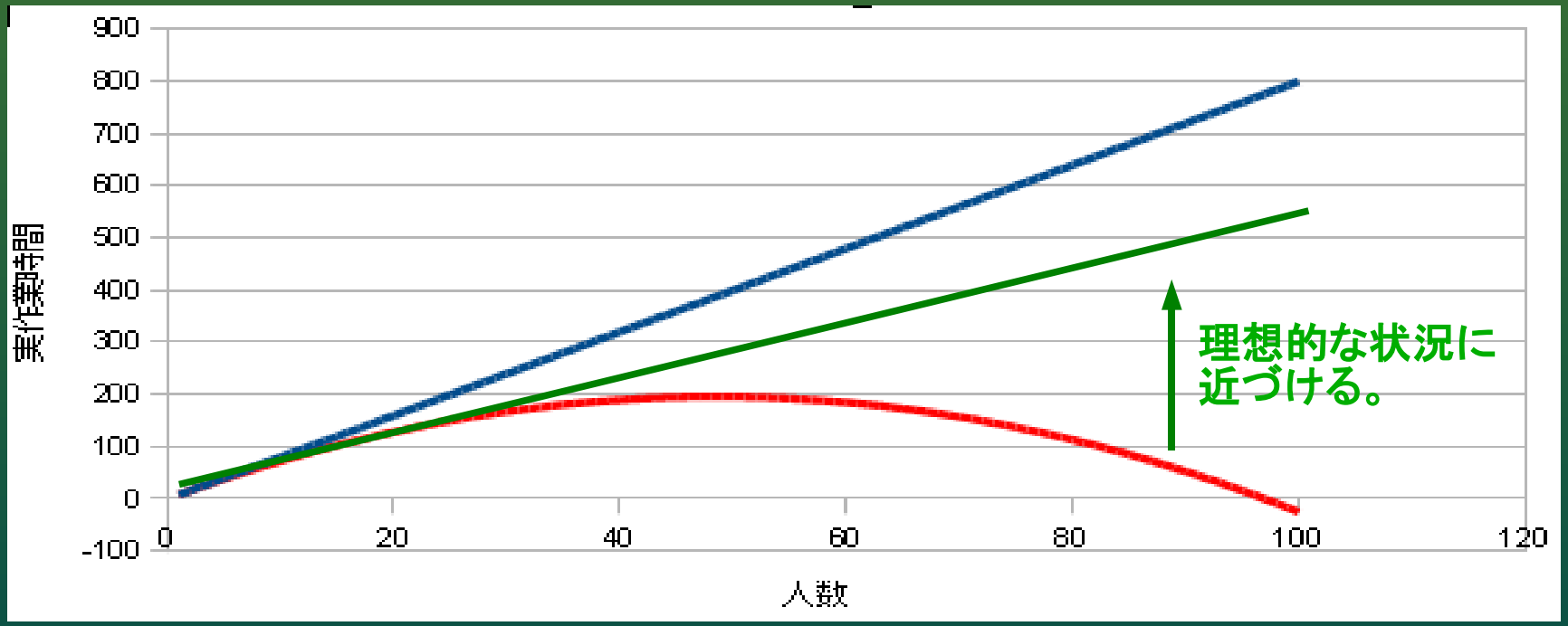
最悪な場合の実作業時間

- 人員の時間の総和 ー コミュニケーションの総和
  - (例) 1 日 = 8 時間、1 組当たりの平均コミュニケーション時間 = 10 分



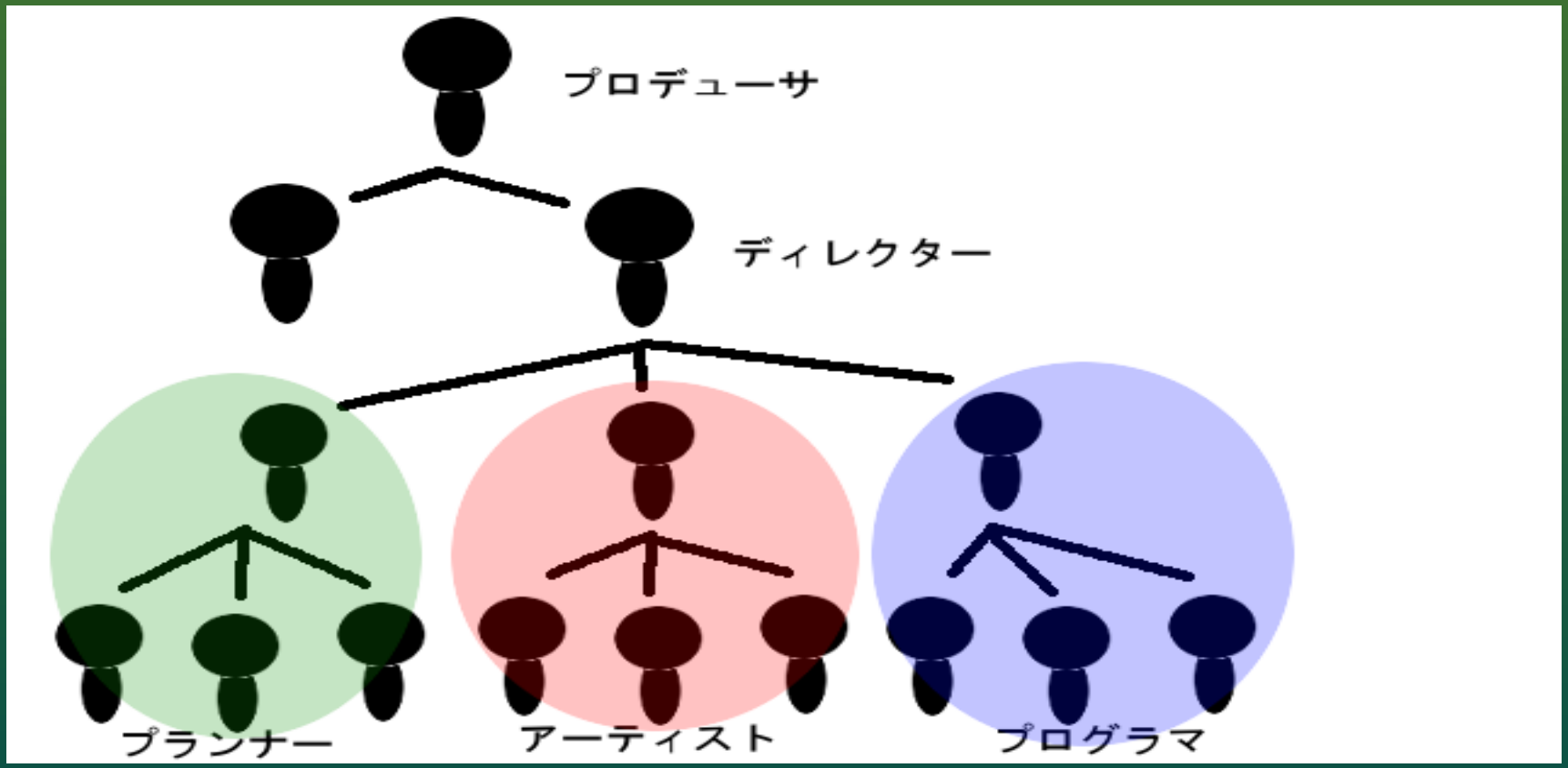
コミュニケーションコストを減らす

- 不要な相互コミュニケーションを削る
  - 階層構造を形成する
    - コミュニケーションの組み合わせは n-1 本に減らせる。



ゲームソフト開発の組織図

- 職種で階層構造を作ると「リードプログラマ」という役割が出現する。



リードプログラマとは何か？

- 職種ツリーのプログラマ系最上位ノードとして成立
  - コミュニケーションを集約する立ち位置
- 役割
  - プログラマを代表しプロジェクトを推進させる。
    - 上位ノード＆兄弟ノードとの関係性から導きだされる。
  - (自分を含む)部下にプロジェクトの状況を踏まえた指示や情報提供を行い、プログラマ系の仕事を完成させる。
    - 下位ノードとの関係性から導きだされる。

多様化するリードプログラマ像

- 状況から求められる「責任」を考える必要がある。
  - 大きいプロジェクトであれば、リードプログラマに相当する役割を 2 名以上で分担する事が必要だろう。
  - 小さいプロジェクトであれば、部下の管理よりはプロジェクト推進に対する責任が重くなるだろう。
- （私見）リードプログラマ的役割に触れる（触れなければならない）プログラマは意外に多い。

具体的なリードプログラマの仕事

- 実際に使っている手法を、プロジェクトの計画段階・実施段階に分け、全部で 10 点紹介します。
  - 想定プロジェクト規模は以下の通りです。
    - ピーク時人員 40 名（プログラマ 10 名）
    - 期間 1 年 6 ヶ月程度
  - 導入にあたっては、状況によりアレンジが必要です。

事前計画系

- プロジェクトの進め方を予定する事により、実作業しやすい環境を作り出す。
  - (1) 大枠での合意形成
  - (2)「組閣」
  - (3) スケジュール作成
  - (4) リソース管理計画

(1) 大枠での合意形成

- これから作るものについて合意する事。
  - 内容の確認
    - 「やる価値があるのか？」「漏れ抜けが無いか？」etc...
    - 内容に正統性が無いと、合意しにくい。
  - 合意の確保
    - 合意の方法、枠組みはルールを決めておくこと。

(1) 大枠での合意形成

- 合意形成のプロセス例

(1) 大枠での合意形成

- ポイント
  - 内容を具体的に詰める事が目的ではない。合意を形成する事が大事。
  - 全部を把握できるに越した事は無いが、最低限、プログラマとしてのポジショントークができるようにする事。
  - プロダクト全体の事は他のセクションの人と協調により作り上げていけば良い。

(2)「組閣」

- 担当分野と人材のマッチングを行なう。
  - 担当分野のリストアップ
    - 大雑把なスケジュールや設計資料を元に担当分野の選定を行う。

(2)「組閣」

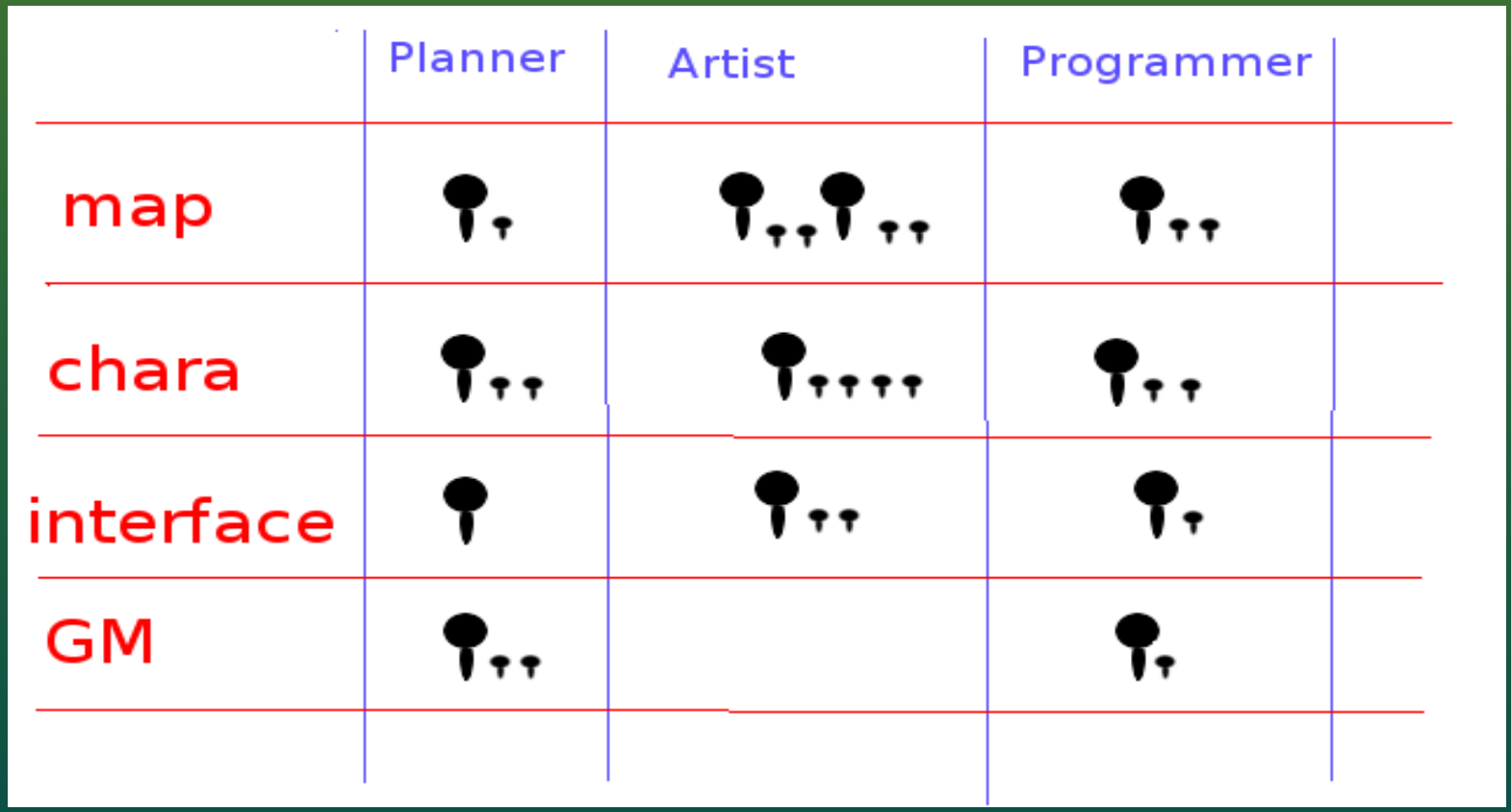
- 荒いスケジュールを元に担当分野を求める例
  - 似たような作業を集めて一つの分野に仕立て上げる。

(2)「組閣」

- ポイント
  - 担当分野別にコア＋サポートメンバという構成になるようにする事。
    - 密結合で複雑度の高い部分を複数人数で実装しようとすると、コミュニケーションコストが高くなる。
  - アサインにあたって…
    - 適材適所を意識する。
    - 他の職種の人の意見も聞いておくと良い。

(2)「組閣」

- 担当分野のリスティングと割り当て例
  - プログラマのセクションが責任範囲



(3) スケジュールの作成

- スケジュールを作成する方法
  - マイルストーンベース
  - タスクリストベース
- 両方を作成しすり合わせて進行計画を作る。

マイルストーンベース

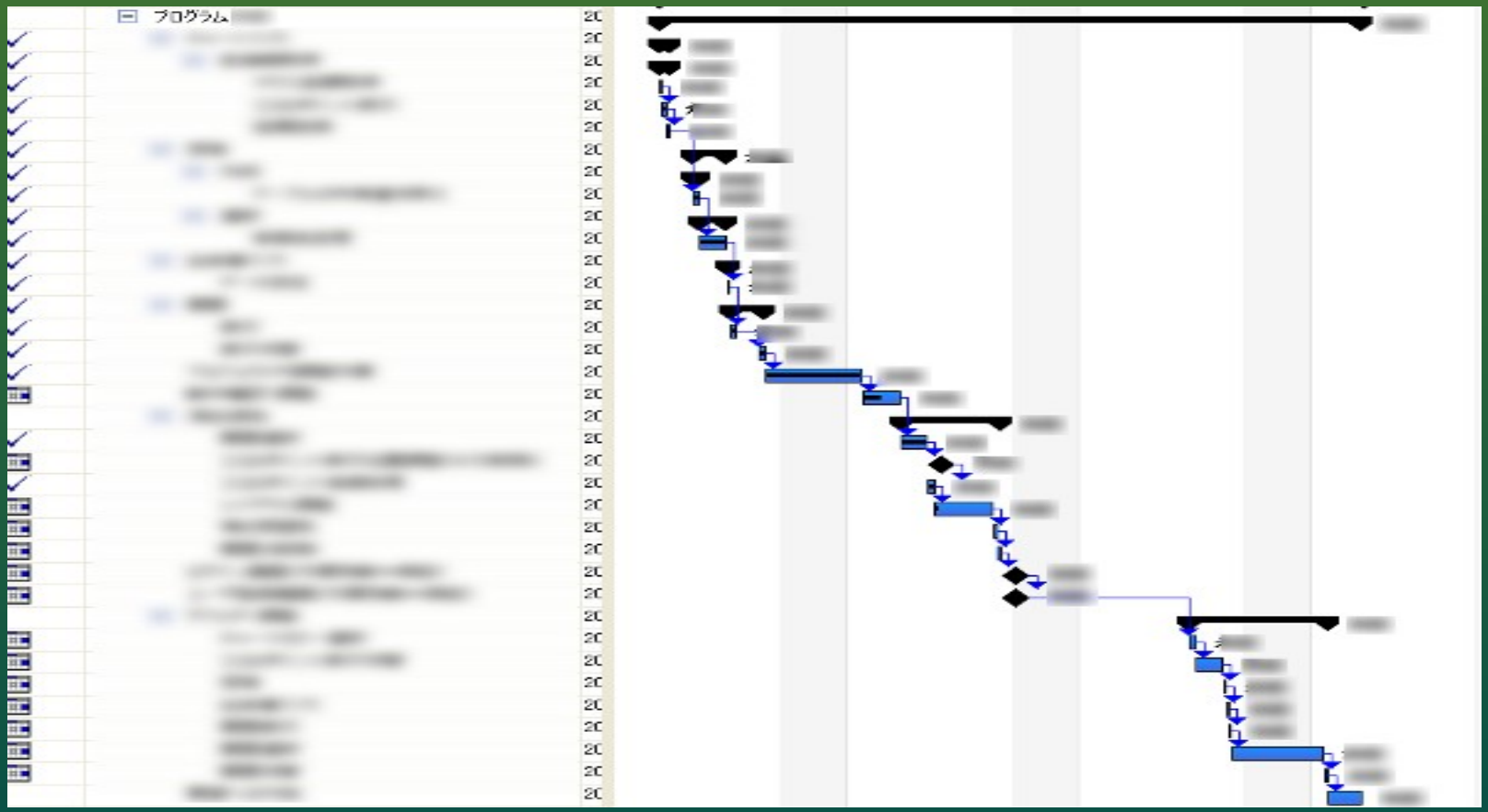
- できるだけ理想論ベースでわかりやすい目標を立てる。
- 期間を月単位等で複数に区切る。
  - (例)
    - 3 月末：応募書類完成
    - 7 月末： 講演メモ作成完了
    - 8 月 15 日 ：プレゼンファイル作成完了
    - 8 月 27 日：最終リハーサル

タスクリストベース

- 作業見積もりの作成
  - タスクリストを作成する
  - 所要時間を予測する
    - 時間単位で作成。(日単位は人によって差が出すぎる)
    - 10 時間を越える項目は細分化可能。
    - 1 日の単位は労働基準法をベースにする。(1日=8時間)
- リードプログラマと実務者双方で相見積もりを取る
  - 抜け漏れの防止

ガントチャートを作成する。

- 縦軸が作業、横軸が日時を示す。
- 横棒が作業時期・期間を示す。



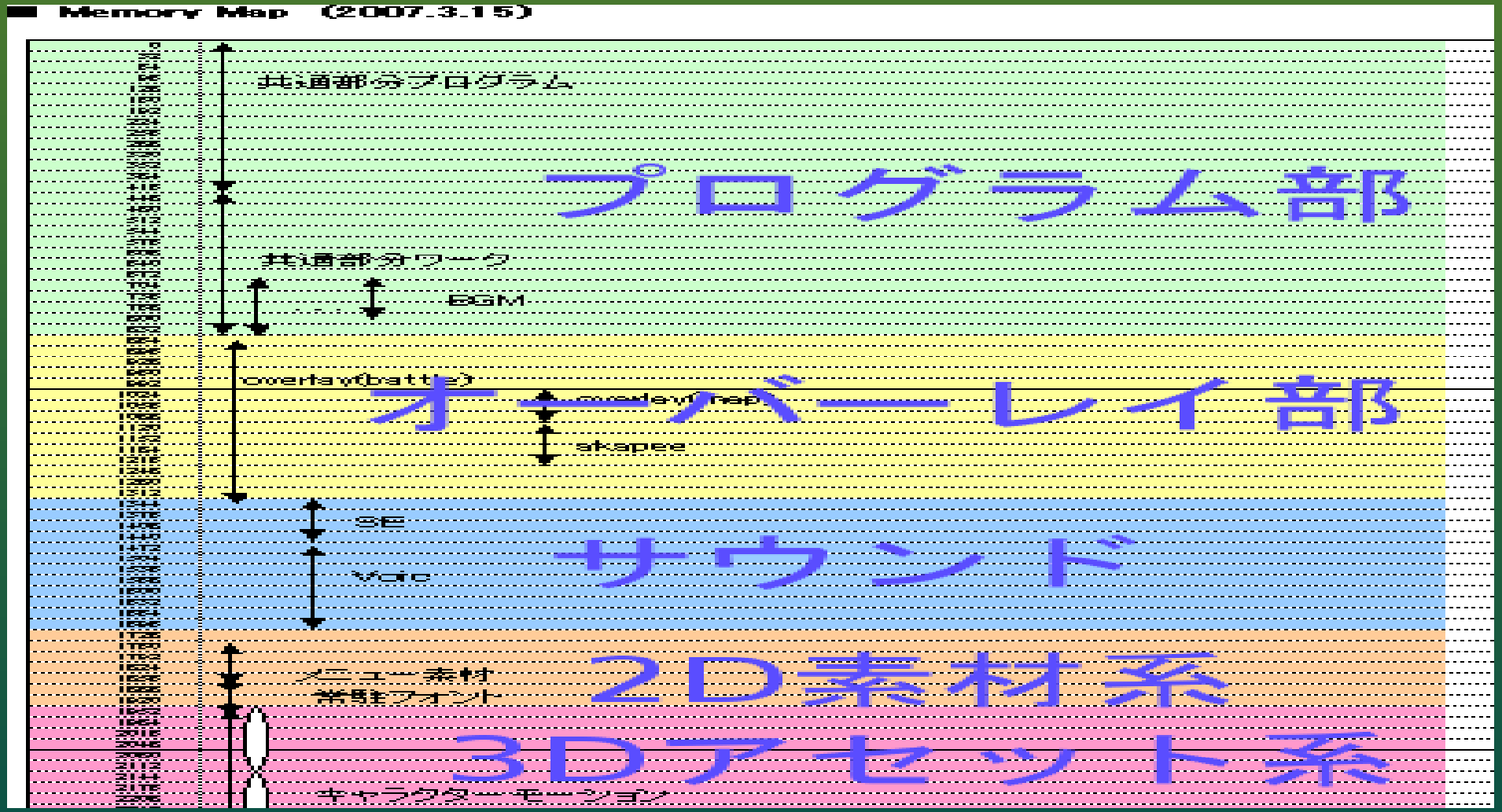
スケジュールの調整

- 終了予想時期がマイルストーン上要求されるタイミングを越える場合の処理
  - 別担当に付けかえる。
  - 効率化ができるかどうか検討する。
  - 作業順を入れ替える。
  - 変更可能ならマイルストーンを変更する。
  - 内容を変更して必要な作業量を低減する。

(4) リソース管理計画

- 全部署にわたって影響を与える実機側リソースは、可視化して管理する事。
  - メモリ（メインメモリ、VRAM等）
  - ストレージ（メディアに入る容量等）

メモリマップ例



(4) リソース管理計画

- ポイント
  - どこに何を割り当てるのかを図表化するとわかり易い。
  - 開発が進むにつれ変化がある事は全然問題は無い。管理されている事が大事。
  - リソース管理が明確になる事により安心してパフォーマンスを追求するための素地が得られる。

制作管理

- 事後的ではあるが、問題を早期に察知して対処する。予防できるものは予防する。
  - (5) デイリーアセスメント
  - (6) 日記を書いてもらう
  - (7) 進行評価
  - (8) プログラマミーティング
  - (9) モジュール別ミーティング

(5) デイリーアセスメント

- 担当部下からの情報収集
  - 毎朝実施する。
  - 1 対 1 でインタビューする。
  - 基本は 2 ～ 3 分程度。問題の解決が必要な場合は、この限りではない。

(5) デイリーアセスメント

- インタビュー内容
  - 前日の作業内容、本日の作業予定を聞いておく。
  - 作業内容に対する問題点を見かけたら指摘する。
  - 作業が一段落したら、動いているところを見せてもらう。
  - たまに当面の見通しも聞く。

(5) デイリーアセスメント

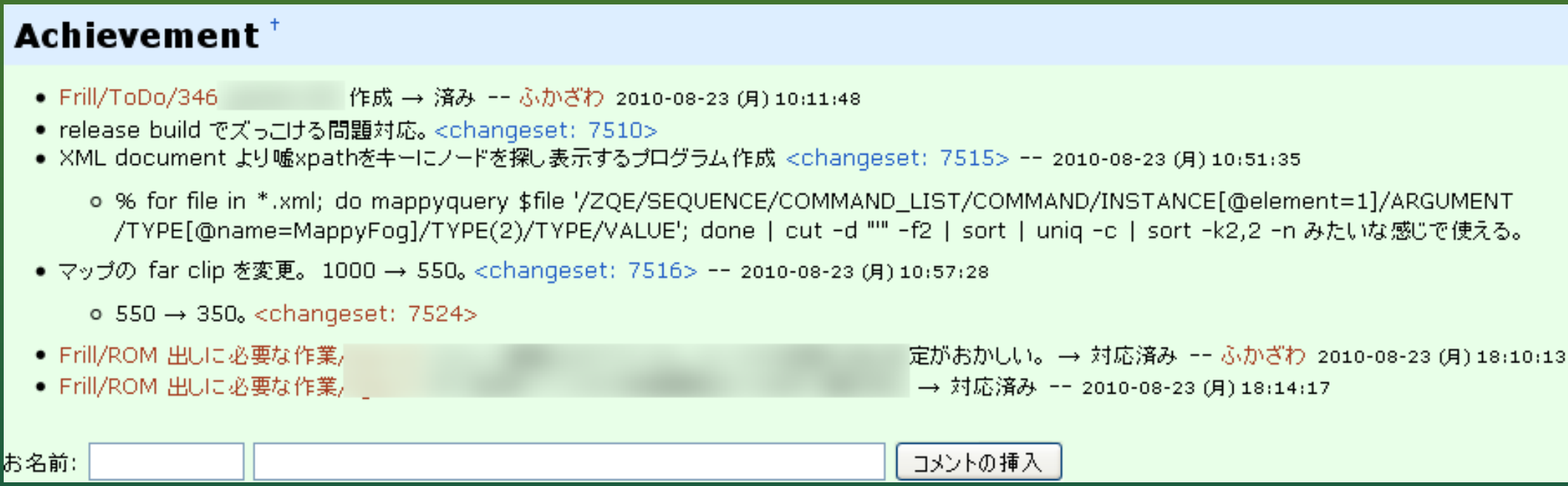
- ポイント
  - 「悩む時間」は実は価値を産んでいない。
    - 答えを教える。（但し、育成との兼ね合いも要考慮）
    - 悩み解決できそうな人を紹介する。
  - スケジュール通りに進める上で日々の遅れ要因を排除する事が重要。
    - 予期しない遅れはかなり減る。

(6) 日記を書いてもらう

- Wiki 等の共有エリアを用意して日記を書いてもらう。
  - 日記は作業概要を箇条書きに書くだけでもよい。
  - ちょっとしたメモでも構わない。

(6) 日記を書いてもらう

- 実例



Achievement †

- Frill/ToDo/346 作成 → 済み -- ふかざわ 2010-08-23 (月) 10:11:48
- release build でズっこける問題対応。<changeset: 7510>
- XML document より嘘xpathをキーにノードを探し表示するプログラム作成 <changeset: 7515> -- 2010-08-23 (月) 10:51:35
  - % for file in *.xml; do mappyquery $file '/ZQE/SEQUENCE/COMMAND_LIST/COMMAND/INSTANCE[@element=1]/ARGUMENT /TYPE[@name=MappyFog]/TYPE(2)/TYPE/VALUE'; done | cut -d "" -f2 | sort | uniq -c | sort -k2,2 -n みたいな感じで使える。
- マップの far clip を変更。 1000 → 550。<changeset: 7516> -- 2010-08-23 (月) 10:57:28
  - 550 → 350。<changeset: 7524>
- Frill/ROM 出しに必要な作業/ 定がおかしい。 → 対応済み -- ふかざわ 2010-08-23 (月) 18:10:13
- Frill/ROM 出しに必要な作業/ → 対応済み -- 2010-08-23 (月) 18:14:17

お名前: コメントの挿入

(6) 日記を書いてもらう

- ポイント
  - アンテナを高くするための方法の一つ。
  - 人に日記を書かせるには、自分が率先して日記を書くとよい。
    - 日記に救われた姿を見せると効果的

(7) 進行評価

- 手順
  - 作業の進行具合を定期的に数値化。
    - ○ 日先行
    - ● 日遅れ
  - 必要に応じて作業割り振りを調整する。
    - 規定以上遅れているのなら担当振り替えを行う
    - 規定以上すすんでいるのなら原点を改める
    - 作業がタスクリストと離れすぎな時はタスクリストを再作成。
  - 割り振りで解決できない場合は、仕様の見直しによる作業量低減や人員投入等を検討する。

(7) 進行評価

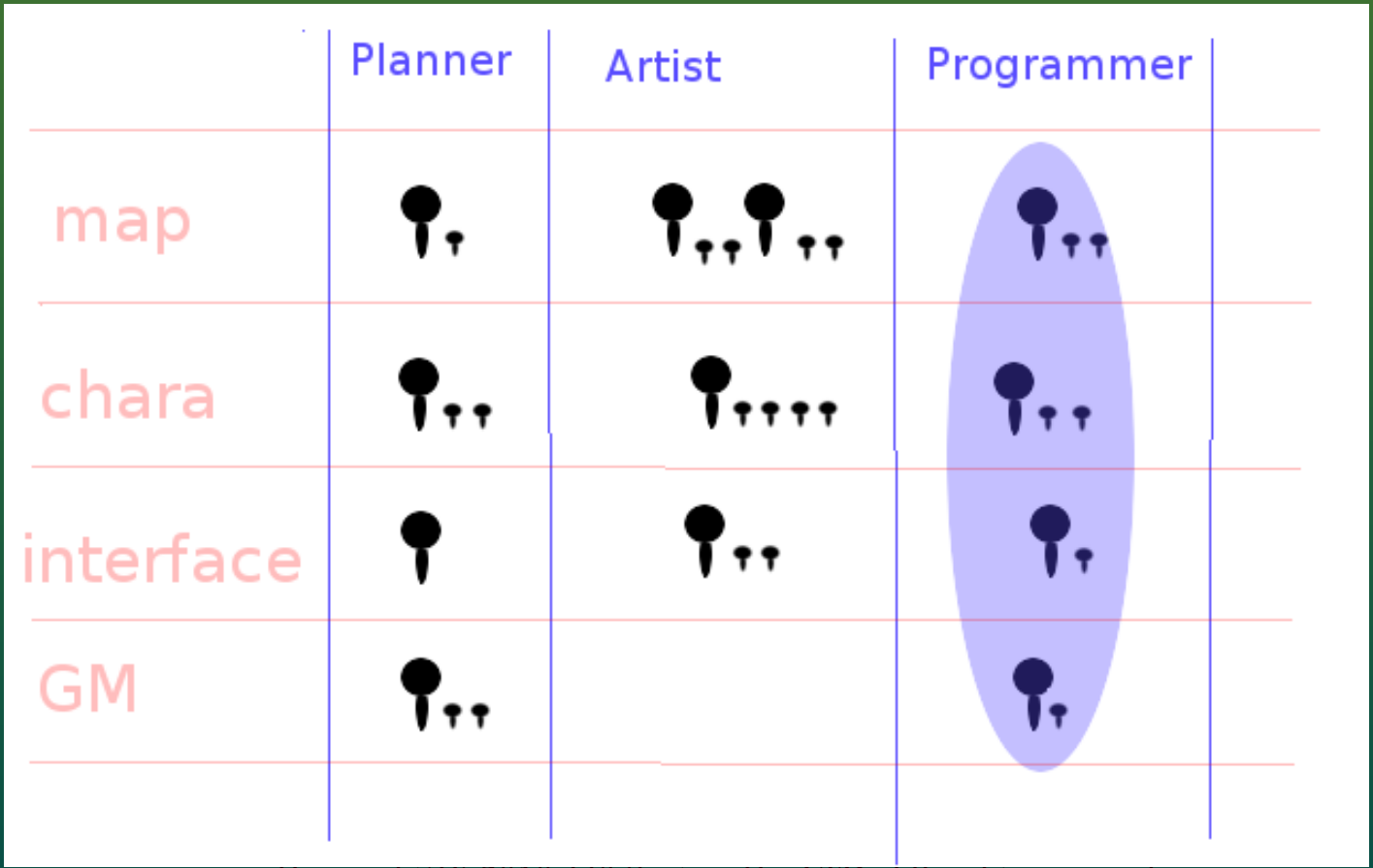
- ポイント
  - 未来分のガントチャートは予想に過ぎない事に注意。
  - 終了予想日が要求を満たすようにコントロールする。
  - 終盤になったら ToDo リストに移行したほうがよい。

ミーティング

- 進行面での合意形成を実施するのに使う。
  - 計画段階で見えていない事が、事前の合意形成の前提を崩すことになる。
  - 人間が完全に未来を見通せない以上、避けられない。
- 改めて合意形成する事によって、不要作業の発生を抑える。
- 目的を明確にする
  - ミーティングで暇しそうな人は呼ばない。
  - 定例化しても、不要ならやらない。

(8) プログラマ･ミーティング

- プログラマとしての意思統一を図るためのミーティング

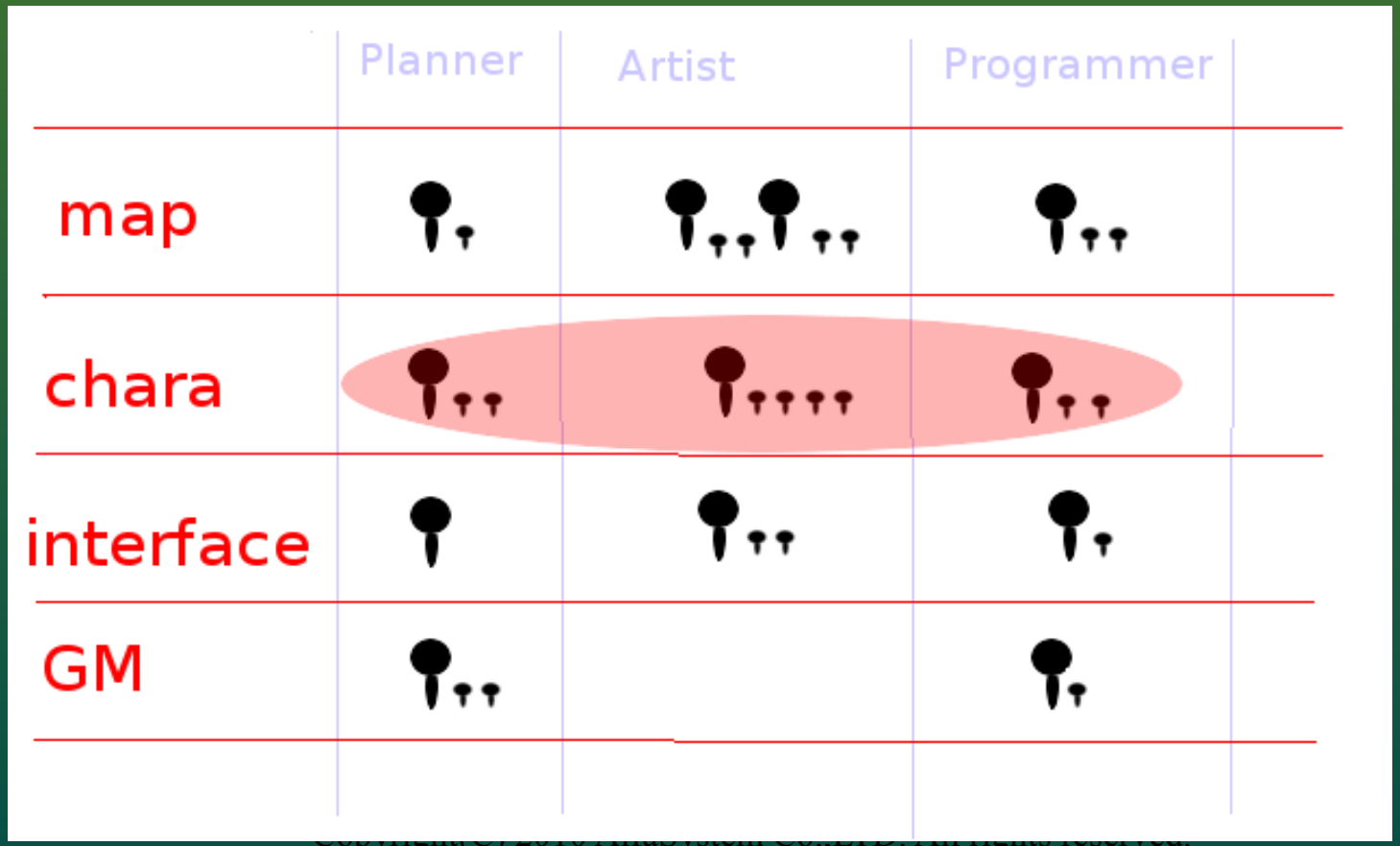


(8) プログラマ・ミーティング

- 不一致が問題になる情報を告知するために行う。
  - プロジェクトとしての決定事項のうち、全員が知っておくべき、もしくは知っておいたほうがよい情報
  - プログラム全体に影響がある問題の存在、解決方法の告知および討議を行う。

(9) モジュール別ミーティング

- モジュール担当者間（職種を問わない）で意思統一を図るためのミーティング



(9) モジュール別ミーティング

- 打ち合わせ内容
  - 担当モジュールの現状把握、問題点、予定を話す。
- ポイント
  - プロダクトの内容、方針に強い影響を及ぼします。
  - 現場で積極的に話をしていたりして順調なようでいても、落ちついて全員で話してみると割と問題点は出てくるから潰せるものは潰しておく事。
  - あたりまえと思うような所でズレが無いか確認したり、今後の予定についてのコンセンサスを得る事が重要。

役割を最大限に果すために

- リードプログラマの役割を果すための時間を最大限確保するようにする。
  - (10) 自分自身は実装作業をしない

(10) 自分自身は実装作業をしない

- なるべく自分の実装作業は外に振りましょう。
  - 基本は部下に作業を振る。
  - 自分の持ち作業も、誰にでもできるように転換する。

(10) 自分自身は実装作業をしない

- (例)ドキュメント化
  - ブランチ作成手順のドキュメント化

注意 [+]

- ブランチ名を XXXXXX とします。
- ブランチを作成する際は、コンパイルをしていない、かつ、データも落としていないプレインな環境を用意しておく事。

手順 [+]

太字になっている部分が実作業部分。一部無駄な手順もありますが、安全性を重視しての事です。

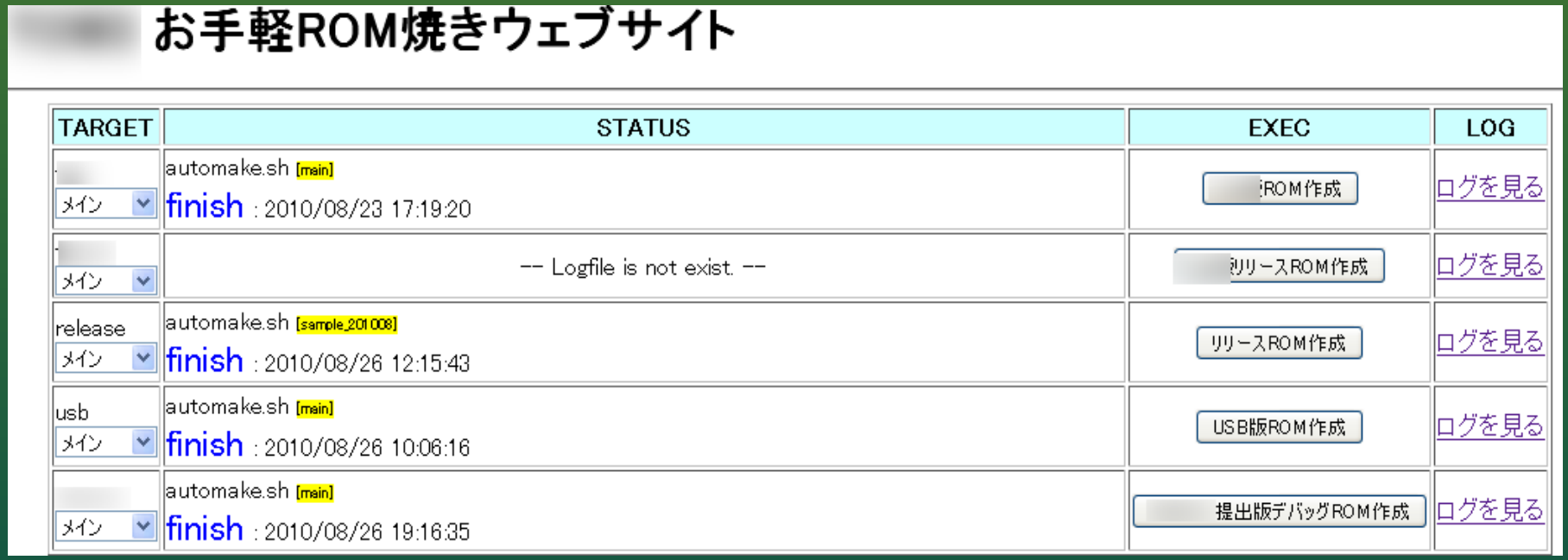
- データを複製
  - [redacted] にログイン
  - **以下を実行。**

```
% cd /resource/[redacted]programmer
% mkdir backup/XXXXXX
% cp -r data backup/XXXXXX
% chmod a+x backup/XXXXXX
```

- レポジトリ処理
  - ブランチの作成

(10) 自分自身は実装作業をしない

- （例）ツールの整備
  - ROM作成プロセスの自動化等



(10) 自分自身は実装作業をしない

- どうしても作業をするなら、時間割を組んで管理業務と開発業務のケジメをつけるようにする。

		月	火	水	木	金
Ⅰ	9:15~10:00	ﾃﾞｲﾘｰｱｾｽﾒﾝﾄ	ﾃﾞｲﾘｰｱｾｽﾒﾝﾄ	ﾃﾞｲﾘｰｱｾｽﾒﾝﾄ	ﾃﾞｲﾘｰｱｾｽﾒﾝﾄ	ﾃﾞｲﾘｰｱｾｽﾒﾝﾄ
Ⅱ	10:00~12:00	管理事務→実作業	管理事務→実作業	管理事務→実作業	管理事務→実作業	ﾓｼﾞｭｰﾙ別ﾐｰﾃｨﾝｸﾞ
Ⅲ	13:00~15:00	実作業	実作業	実作業	実作業	ﾓｼﾞｭｰﾙ別ﾐｰﾃｨﾝｸﾞ
Ⅳ	15:15~17:30	実作業	実作業	ﾌﾟﾛｸﾞﾗﾏﾐｰﾃｨﾝｸﾞ	実作業	全体ﾐｰﾃｨﾝｸﾞ
Ⅴ	17:30~18:30	管理事務	管理事務	<ﾉｰ残業ﾃﾞｰ>	管理事務	(ﾌﾟﾛｸﾞﾗﾏ勉強会)

最後にアドバイス

- 問題を曖昧なままに流さない。
  - 問題を突き詰めることが改善を生み出す。
- 「餅は餅屋」
  - エキスパートや部下を活用する。
- 多少の失敗は気にしない
  - 同じミスをしないようにすればいい。
  - 「完成させるための具体的な行動」を常に見せる事

まとめ

- リードプログラマの役割
  - プログラマを代表しプロジェクトを推進させる。
  - (自分を含む)部下にプロジェクトの状況を踏まえた指示や情報提供を行い、プログラマ系の仕事を完成させる。
- 実務例
  - 事前計画
    - 大枠の合意形成、組閣、スケジュール作成、リソース計画
  - 制作管理
    - デイリーアセスメント、日記、進行評価、ミーティング
  - 自分自身は実装作業をしない。

質問タイム

ご清聴ありがとうございました。

- ご意見･ご質問等ありましたら
  - メールで受け付けています。
    - fukazawa@alfasystem.co.jp
  - Twitter やってます。
    - @superfrill
  - アンケートへご記入お願いします。