SQUARE ENIX OPEN CONFERENCE

ゲーム開発 プロジェクトマネジメント講座

２０１１年１０月８日
株式会社スクウェア・エニックス
CTO 橋本 善久

©SQUARE-ENIX 2011

なぜプロジェクトは失敗するのか？

©SQUARE-ENIX 2011

プロジェクトの失敗ポイント

- 見込みより売上が少ない
- 計画よりもコストがかかっている
- 発売時期が遅れた
- 発売に間に合わせるため内容が削られた
- ユーザーの評判が悪い
- 不具合が発生
- スタッフの満足度が低い、故障者が出た、辞めてしまった
- など・・・

©SQUARE-ENIX 2011

プロジェクトの失敗ポイントの分類

- スコープ（コンテンツの範囲）の問題
- 品質の問題
- コストの問題
- 時間の問題
- リソース（人員・環境）の問題
- ビジネスの問題

©SQUARE-ENIX 2011

プロジェクトの失敗ポイントの分類

- スコープ（コンテンツの範囲）の問題
- 品質の問題
- コストの問題
- 時間の問題
- リソース（人員・環境）の問題
- ビジネスの問題

絡み合う複雑なパラメータ

これらをうまく制御したい

©SQUARE-ENIX 2011

問

プロジェクトの初期に立てた計画は最終的にどれほどの誤差を生むのでしょうか？

©SQUARE-ENIX 2011

当初計画比で
１．２倍？
１．５倍？
それとも２倍？

©SQUARE-ENIX 2011

いえいえ
そんな甘い数値では
ないのです

©SQUARE-ENIX 2011

まずは計画を変動させる
主要因を列挙してみましょう

©SQUARE-ENIX 2011

計画の主な変動要因

- **スコープの変化**
  - 仕様の追加・変更
- **タスク分解のエラー**
  - 既知の仕様に対してタスクの洗い出し漏れがあった場合
- **見積もりのエラー**
  - タスク消化にかかる時間を見誤った場合
- **一日あたりの作業時間見込みエラー**
  - 会議その他で思ったよりも作業時間が取れなかった場合など
- **人員計画のエラー**
  - 思ったより人数を増やせない
  - 思ったより能力がない
  - 辞められる、故障者が出る
- **品質マネジメントのエラー**
  - 品質が上がらないので作り直し
- **技術マネジメントのエラー**
  - プログラム的に実現が困難なので作り直し
- **その他いろいろ・・・**

©SQUARE-ENIX 2011

変動要因別の変化幅の例

- 仕様追加でスコープ変更⇒１．５倍
- タスク分解のエラー⇒１．４倍
- 見積もりのエラー⇒１．３倍
- 一日あたりの作業時間見込みエラー
  ⇒作業時間－２０％＝１．２５倍
- 人員計画のエラー
  - 思ったように増員できなかった⇒－２０％＝１．２５倍
  - 思ったような能力が無かった⇒－３０％＝１．４倍
- 品質マネジメントのエラー⇒１．３倍
- 技術マネジメントのエラー⇒１．３倍
- ・・・

©SQUARE-ENIX 2011

それぞれの要因による計画増加分を掛け合わせてみると・・・

※それぞれの要因は独立しており、おおよそ直交性があると仮定します

©SQUARE-ENIX 2011

$$1.5 \times 1.4 \times 1.3 \times 1.25 \times 1.25 \times 1.4 \times 1.3 \times 1.3$$

©SQUARE-ENIX 2011

1.5×1.4×1.3×1.25
×1.25×1.4×1.3×1.3

≒10倍！！

©SQUARE-ENIX 2011

まさかの
当初計画比１０倍という
無茶苦茶な数値が
出てきました

©SQUARE-ENIX 2011

相当ラフな計算ではありますが、
如何に直感と異なる大きな誤差
になるのかはお分かりになれたか
と思います

※あくまでも、直感よりも遥かに大きいということを伝えるための計算なので、その計算方法や数値の精度に対しては目をつむってください

©SQUARE-ENIX 2011

ではこの途方もない誤差に対してみなさんは普段どのように対処しているのでしょうか

©SQUARE-ENIX 2011

１０倍なんて膨らんだら
解決は無理？
いえいえ結構みなさん日常的に
対処してるのです

©SQUARE-ENIX 2011

対処1

仕様を３５％削減

©SQUARE-ENIX 2011

対処1

仕様を３５％削減
⇒１．５４倍の対策効果

©SQUARE-ENIX 2011

対処2

人員を60％追加投入

©SQUARE-ENIX 2011

対処２

人員を６０％追加投入
⇒１．６倍の対策効果

※もちろん人を足したからといって素直に人数分の効果が
出るわけではありませんが簡易計算として割り切ります

©SQUARE-ENIX 2011

対処3

期間を50％延長

©SQUARE-ENIX 2011

対処３

期間を５０％延長
⇒１．５倍の対策効果

©SQUARE-ENIX 2011

対処4

品質を30％妥協する

©SQUARE-ENIX 2011

対処4

品質を30％妥協する
⇒1．43倍の対策効果

※品質を下げる事で工数を減らすという意味で単純計算しています

©SQUARE-ENIX 2011

1.54×1.6×1.5×1.43

©SQUARE-ENIX 2011

１．５４×１．６×１．５×１．４３
≒５．６４倍の対策効果

©SQUARE-ENIX 2011

１．５４×１．６×１．５×１．４３
≒５．６４倍の対策効果

うーんまだまだ
１０倍の対策効果まで足りません
これでは発売不能です

©SQUARE-ENIX 2011

なにか良い手だては無いでしょうか

©SQUARE-ENIX 2011

あ、

©SQUARE-ENIX 2011

ひとつ効果的な方法が残っていました

©SQUARE-ENIX 2011

対処５

月の労働時間を
１６０時間から
２９０時間にする

©SQUARE-ENIX 2011

対処５

月の労働時間を
１６０時間から
２９０時間にする
⇒１．８倍の対策効果

©SQUARE-ENIX 2011

1.54×1.6×1.5×1.43×1.8

©SQUARE-ENIX 2011

１．５４×１．６×１．５×１．４３×１．８

≒１０倍の対策効果

©SQUARE-ENIX 2011

１．５４×１．６×１．５×１．４３
×１．８
≒１０倍の対策効果

これで晴れて対策を
打つことができました！！
発売可能ですね！！！

©SQUARE-ENIX 2011

めでたしめでたし！！

©SQUARE-ENIX 2011

めでたしめでたし！！

・・・のはずもありませんよね

©SQUARE-ENIX 2011

対処１　仕様を３５％削減
対処２　人員を６０％追加投入
対処３　期間を５０％延長
対処４　品質を３０％妥協する
対処５　月の労働時間を８０％増やす

©SQUARE-ENIX 2011

対処１　仕様を３５％削減
対処２　人員を６０％追加投入
対処３　期間を５０％延長
対処４　品質を３０％妥協する
対処５　月の労働時間を８０％増やす

見事なデスマーチの完成です
結構ありがちな風景ではないでしょうか

©SQUARE-ENIX 2011

プロジェクトの失敗ポイントの分類

- スコープ（コンテンツの範囲）の問題
- 品質の問題
- コストの問題
- 時間の問題
- リソース（人員・環境）の問題
- ビジネスの問題

©SQUARE-ENIX 2011

プロジェクトの失敗ポイントの分類

- スコープ（コンテンツの範囲）の問題
- 品質の問題
- コストの問題
- 時間の問題
- リソース（人員・環境）の問題
- ビジネスの問題

がんばって着地させたが・・・

プロジェクトとしては失敗

©SQUARE-ENIX 2011

プロジェクトの失敗ポイントの分類

- スコープ（コンテンツの範囲）の問題
- 品質の問題
- コストの問題
- 時間の問題
- リソース（人員・環境）の問題
- ビジネスの問題

ビジネスとして成功か失敗かは発売するまでは不明

がんばって着地させたが・・・

プロジェクトとしては失敗

©SQUARE-ENIX 2011

プロジェクトの失敗ポイントの分類

- スコープ（コンテンツの範囲）の問題
- 品質の問題
- コストの問題
- 時間の問題
- リソース（人員・環境）の問題
- ビジネスの問題

この状況で売れるか売れないか
どっちが幸福かは難しいところ

ビジネスとして成功か失敗か
は発売するまでは不明

がんばって着地させたが・・・

プロジェクトとしては失敗

©SQUARE-ENIX 2011

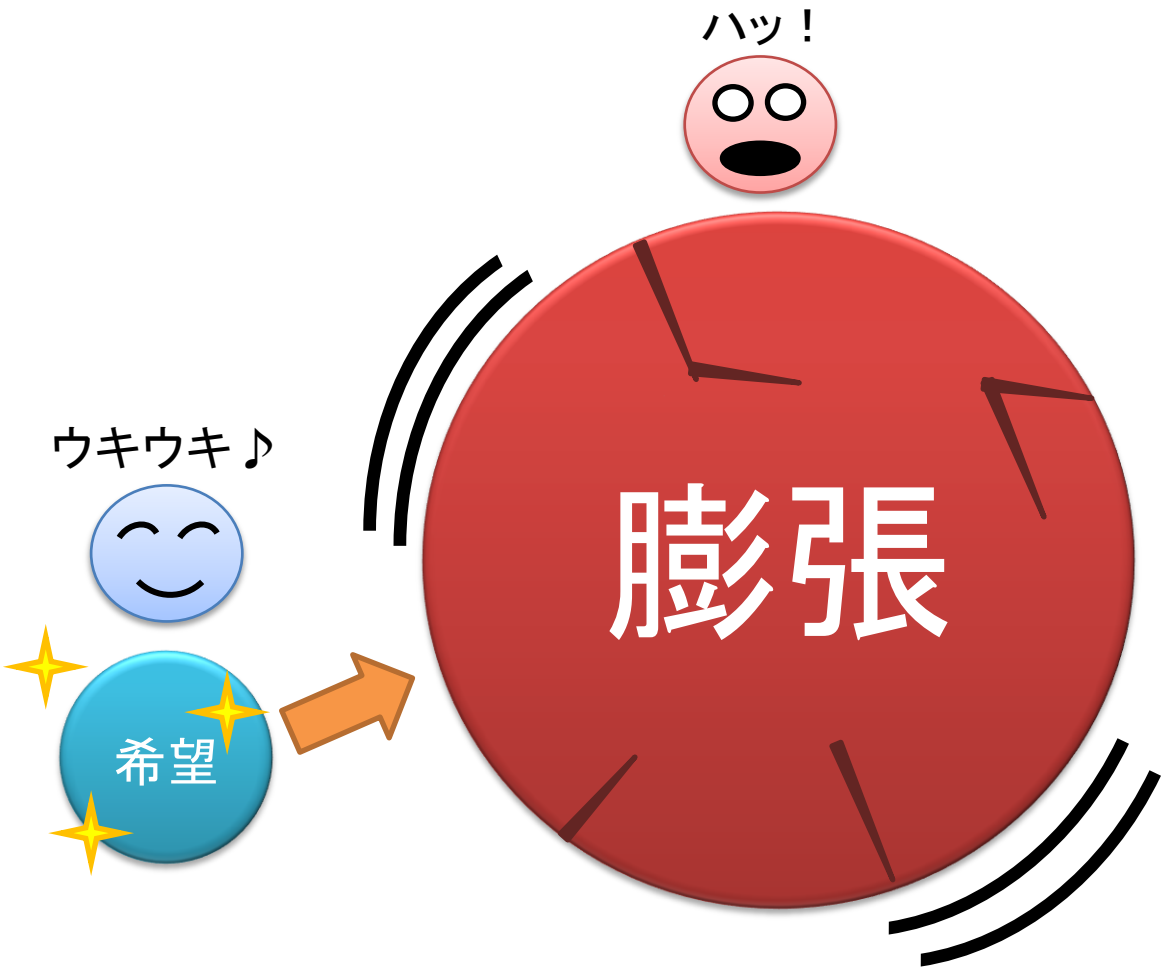
一般的に、ソフトウェア開発ではとてつもなく巨大な計画の誤差が生まれ、それに対して途方もない苦労を割いてどうにかこうにか対処し、不満足な状態で無理やり着地させているのが実情です。

※もちろんうまく行っているプロジェクトにはあてはまりません

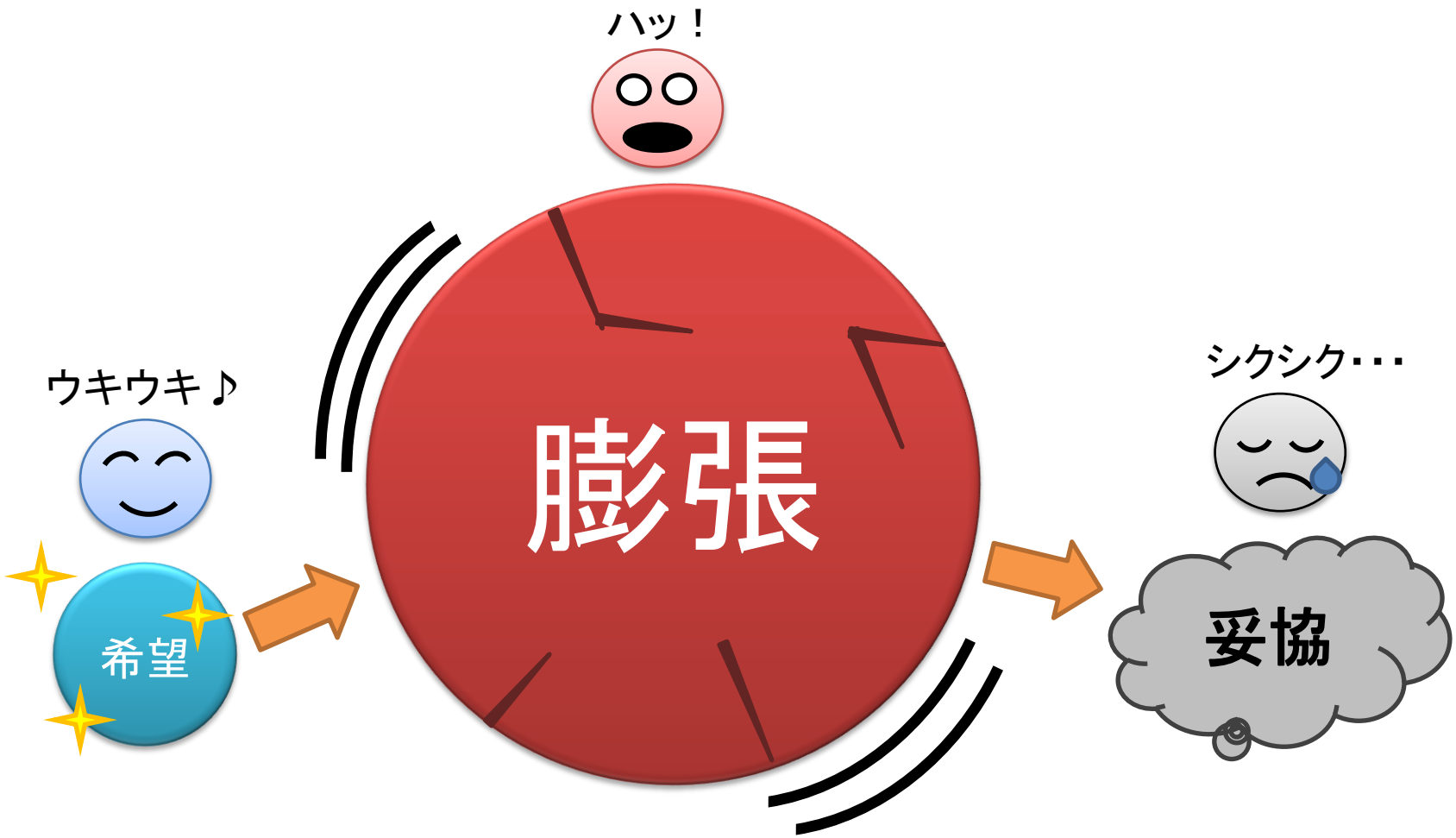
©SQUARE-ENIX 2011

ウキウキ♪

©SQUARE-ENIX 2011

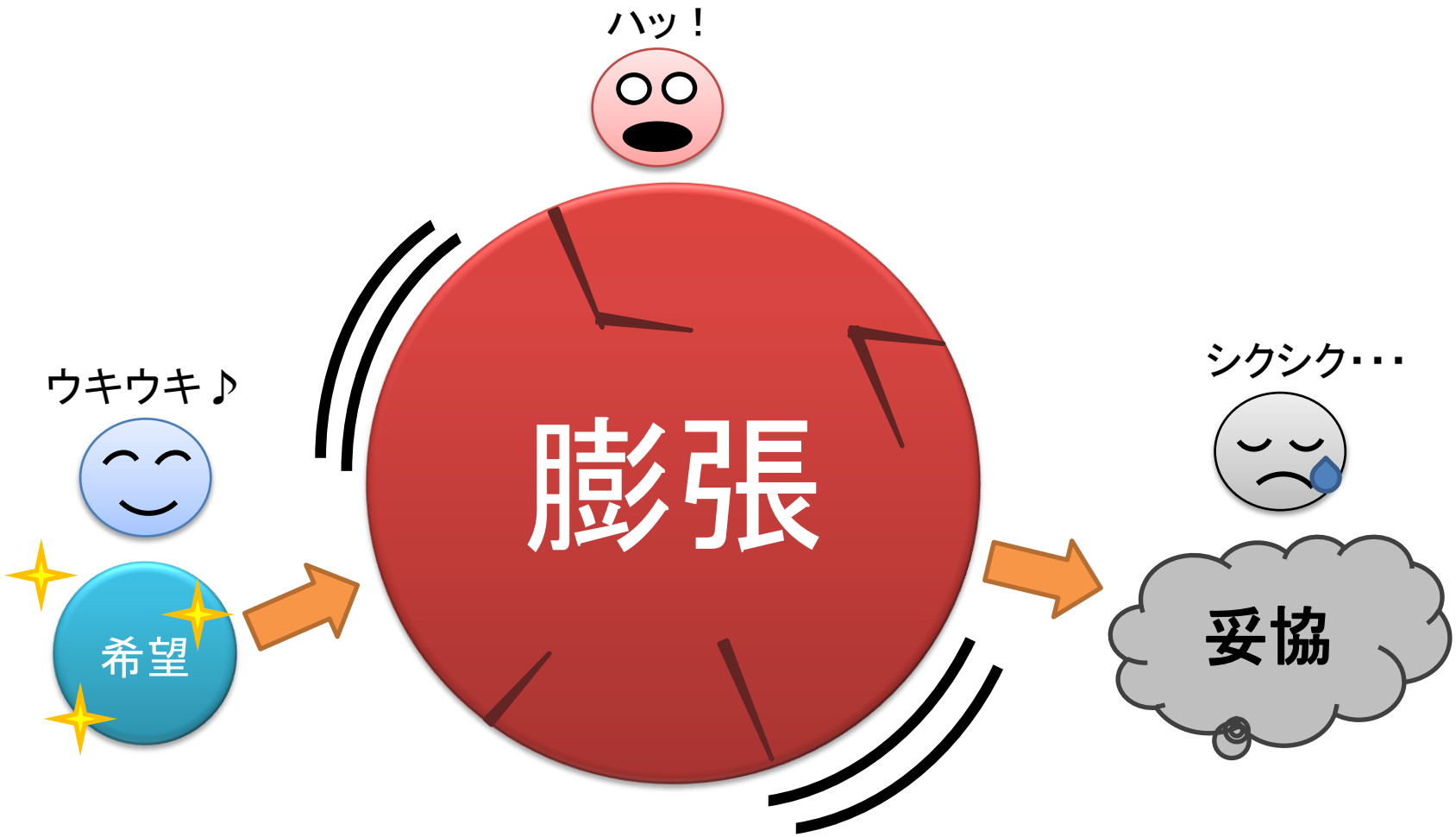


©SQUARE-ENIX 2011



©SQUARE-ENIX 2011

失敗プロジェクト



©SQUARE-ENIX 2011

プロジェクトの初期計画とその結果の誤差は１０倍になってもおかしくないほどに乖離してしまうのです

※現実には初期計画誤差１００倍などというトンデモケースもありえると考えます

©SQUARE-ENIX 2011

つまり

©SQUARE-ENIX 2011

プロジェクトの
正確な予想なんて
出来っこないのです

©SQUARE-ENIX 2011

では、プロジェクトを制御することは無理なのでしょうか？やるだけ無駄なのでしょうか？

©SQUARE-ENIX 2011

いいえ、
プロジェクトを制御することは
可能です

©SQUARE-ENIX 2011

予想は難しくても
制御はできるのです

©SQUARE-ENIX 2011

プロジェクトとは
不確実なものである

という事実を受け入れ、
問題が大きくなってから対応する
行き当たりばったりの運営から

用意周到な事前対処と
積極的な事後対処

を実践することでプロジェクトを制御する
ことが可能になります

©SQUARE-ENIX 2011

不確実性を乗りこなす

- **事前対策**（調査・戦略・設計・計画）
  - **不確実性を減らす**
    - 見通しが悪い部分の調査や検証/実験をしっかりと行う
    - 作業データを取り、フィードバックをかけて予測精度を向上させる
    - 計画規模を小さくする
      - どうせ計画は膨らむのだから最初に目いっぱいコンパクトにする
    - 設計をなるべく詳細に行う
    - リスクの高い要素を予め極力減らす
  - **不確実でも良い事にする**
    - リスクの高い要素を無くしてても成立する設計にする
      - その要素が無くなっても商品の価値が下がらない設計にする
        - » **トカゲのしっぽ**としていつでも切れるようにする
- **事後対策**（イテレーション）
  - **変化や問題発生にすばやく気づく**
  - **変化や問題にすばやく対応する**

©SQUARE-ENIX 2011

まずはイテレーションの大切さから

©SQUARE-ENIX 2011

Plan-Do-Checkサイクル

©SQUARE-ENIX 2011

ん？
という方いますよね。

©SQUARE-ENIX 2011

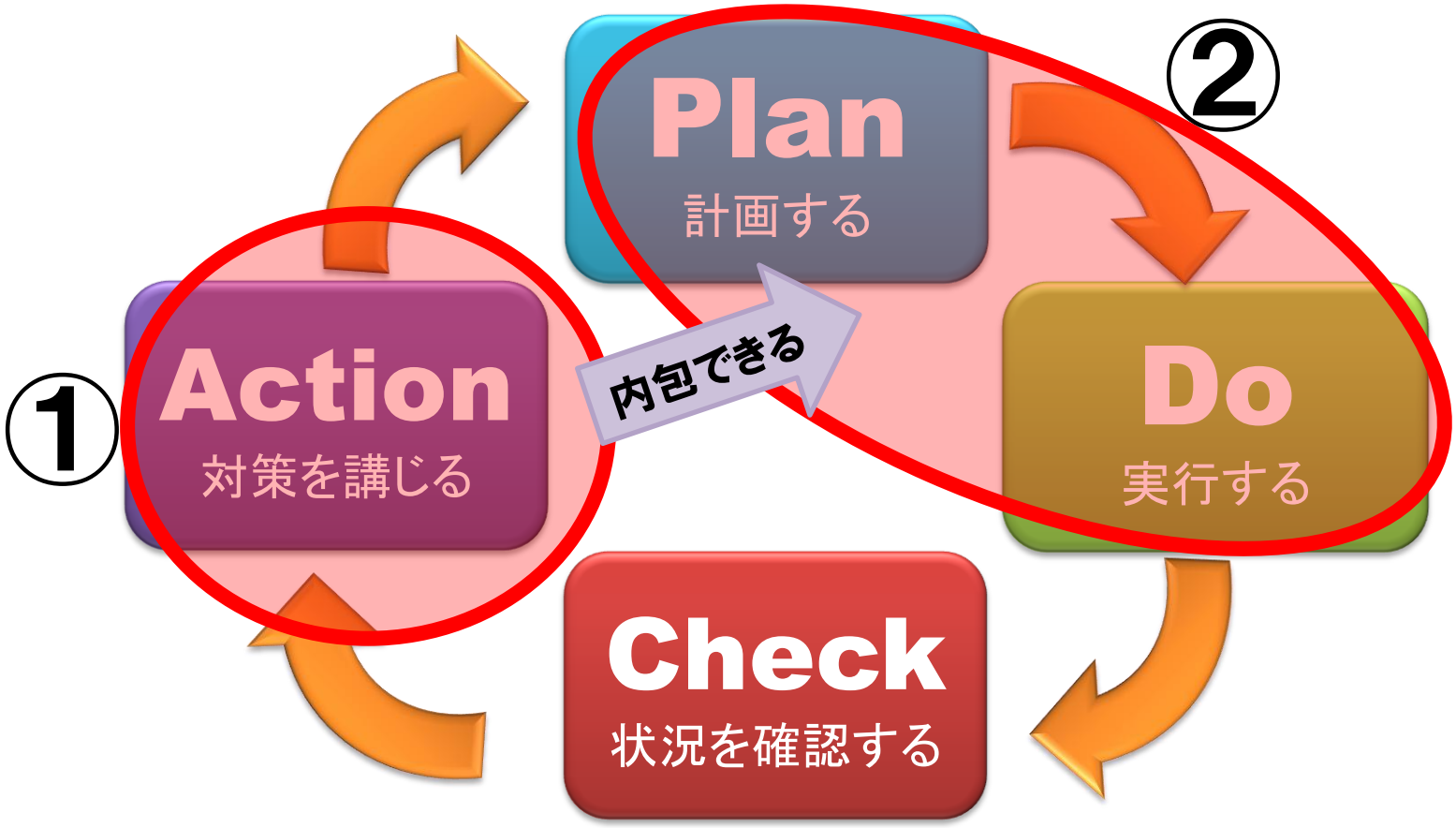
ちょっと補足

©SQUARE-ENIX 2011

世間ではPDCAという言い方が主流ですが

©SQUARE-ENIX 2011

①は②に内包できて冗長であり
どうにも違和感があるので
この表現は本講義では用いません



©SQUARE-ENIX 2011

戻ります

©SQUARE-ENIX 2011

Plan-Do-Checkサイクル

©SQUARE-ENIX 2011

ありがちなプロジェクト

問題が無視できないほどに大きくなって
仕方なく計画を途中で見直している

©SQUARE-ENIX 2011

ダメなプロジェクト

一回計画を立ててあとは作りつづけるのみ。
リスクは増大する一方

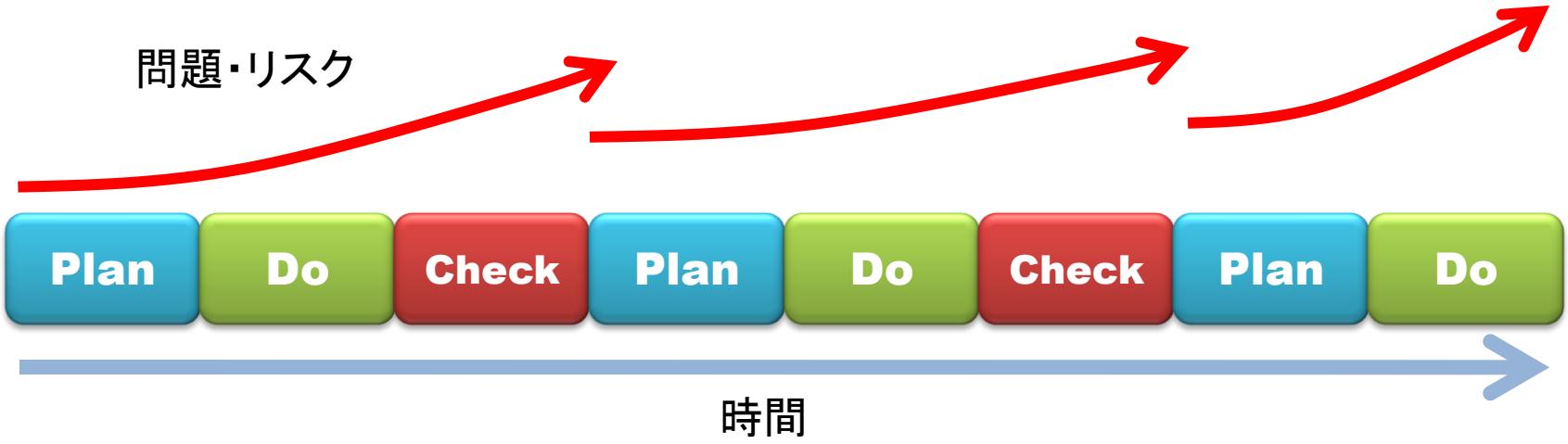
©SQUARE-ENIX 2011

ダメダメなプロジェクト

計画もなくただやみくもに作る。
プロジェクト初期から問題だらけ。

©SQUARE-ENIX 2011

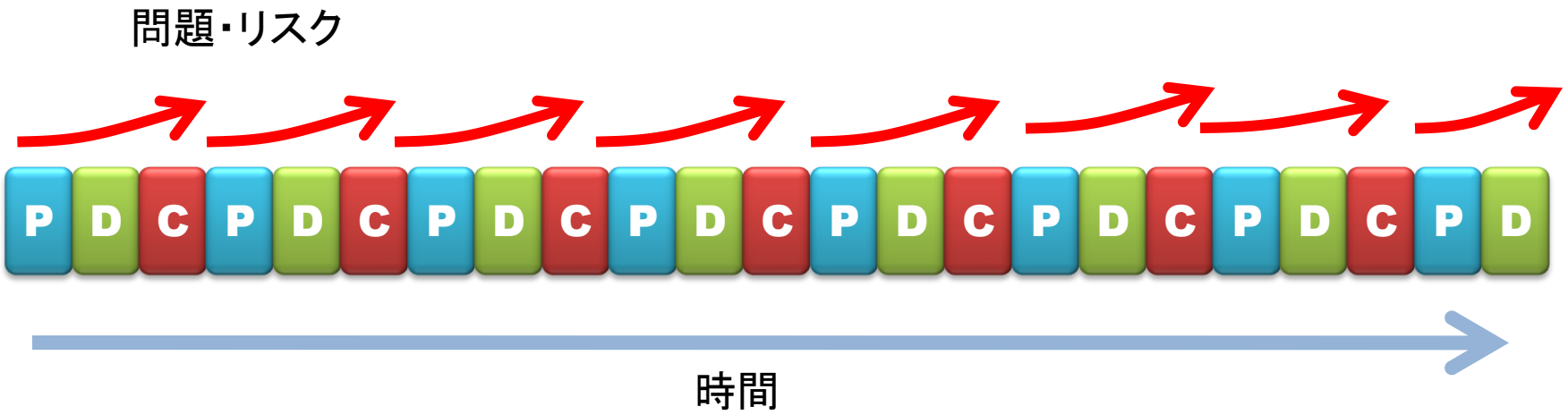
惜しいプロジェクト



計画⇒実行⇒振り返り⇒再計画⇒実行⇒振り返り⇒再計画・・・
というサイクルを行っているが、その単位が大きい。

©SQUARE-ENIX 2011

良いプロジェクト



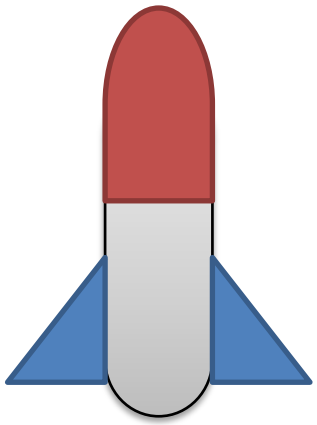
短いサイクルで
計画⇒実行⇒振り返り⇒再計画⇒実行⇒振り返り⇒再計画・・・
と回すことで問題やリスクが早い段階で摘み取られるので、安定したプロジェクト運営が行われる。

©SQUARE-ENIX 2011

ひとつ例を見てみます

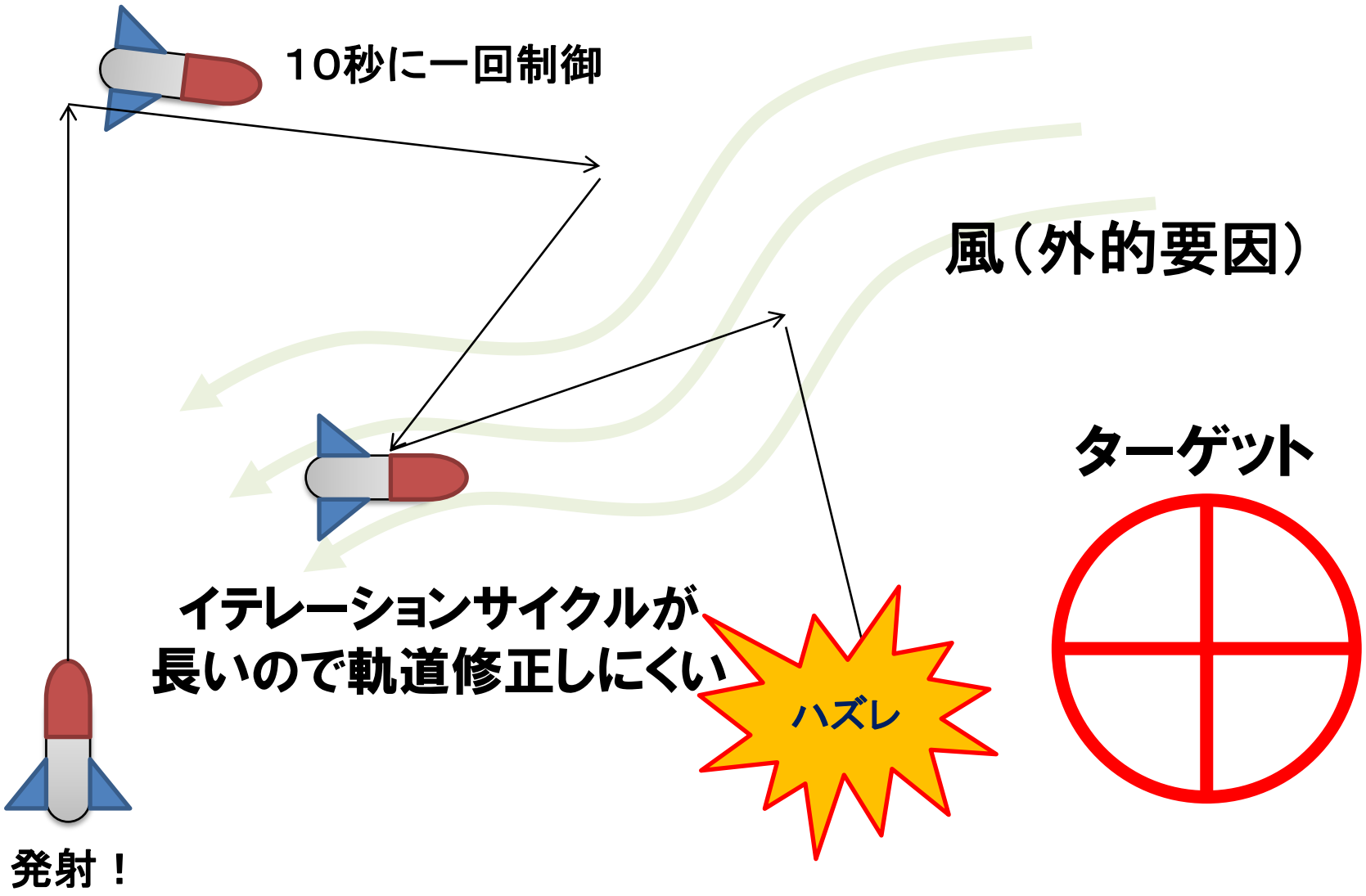
©SQUARE-ENIX 2011

イテレーションの例
ミサイルを飛ばす



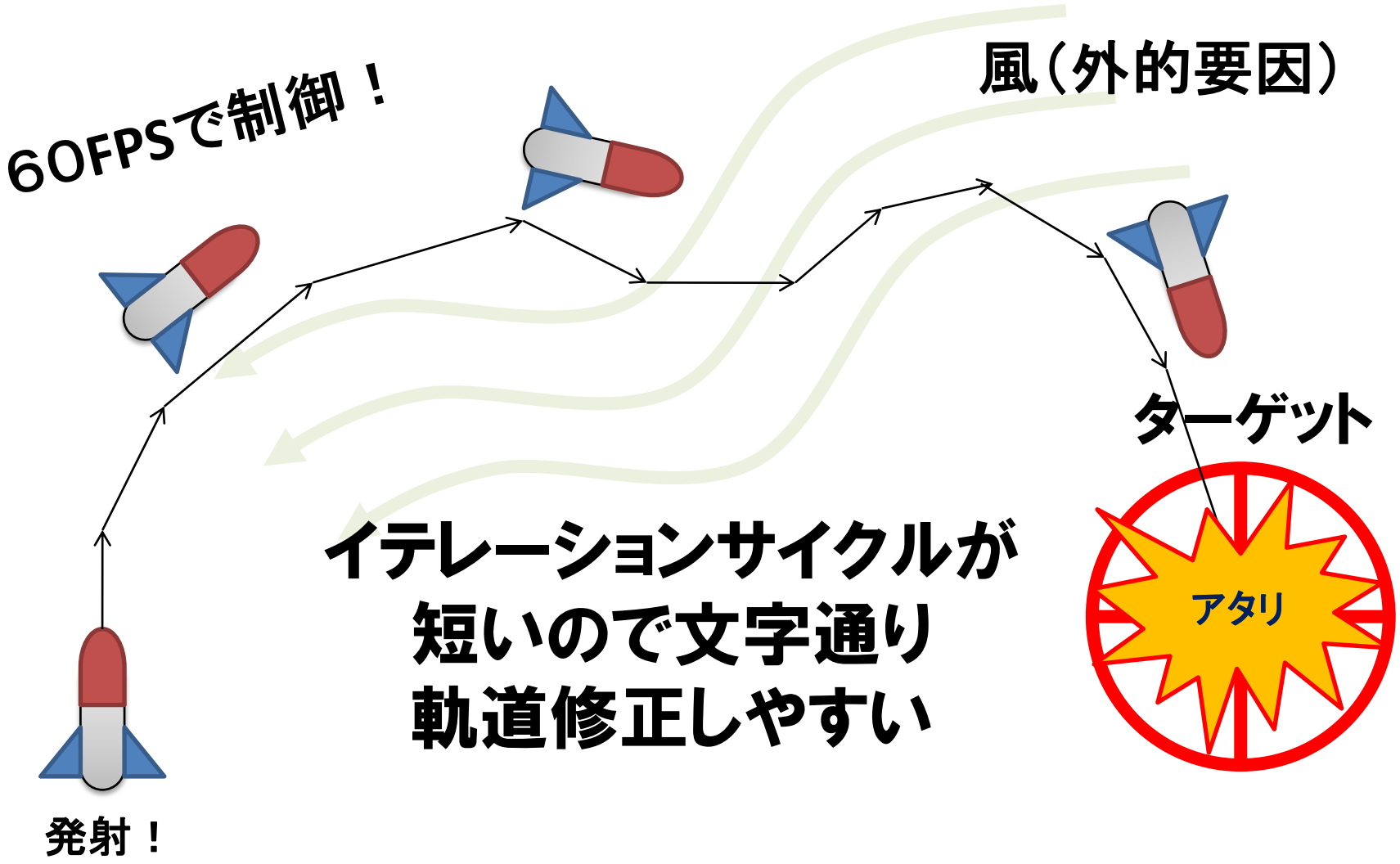
©SQUARE-ENIX 2011

ミサイルを飛ばす（ダメな例）



©SQUARE-ENIX 2011

ミサイルを飛ばす（良い例）



©SQUARE-ENIX 2011

すばらしい

©SQUARE-ENIX 2011

じゃあイテレーションを細かくまわせばプロジェクトマネジメントはバッチリ！

©SQUARE-ENIX 2011

じゃあイテレーションを
細かくまわせばプロジェクト
マネジメントはバッチリ！

・・・でもありません

©SQUARE-ENIX 2011

実はなにも考えずに
細かくイテレーションを
まわしているだけでは
大きな落とし穴にハマります

©SQUARE-ENIX 2011

その話に移る前準備として・・・

©SQUARE-ENIX 2011

近頃はアジャイル
という言葉が
流行っていますが・・・

©SQUARE-ENIX 2011

誤解の中運用されている
ケースもあるように思います

©SQUARE-ENIX 2011

よくあるアジャイルの誤解

- 仕様や設計なんていらない！
- ドキュメントなんていらない！
- 計画なんていらない！
- 「作って、壊して」を短いイテレーションで繰り返せば万事解決！
- ウォーターフォールは悪！

©SQUARE-ENIX 2011

よくあるアジャイルの誤解

- 仕様や設計なんていらない！
- ドキュメントなんていらない！
- 計画なんていらない！
- 「作って、壊して」を短いイテレーションで繰り返せば万事解決！
- ウォーターフォールは悪！

んなわけない！！

©SQUARE-ENIX 2011

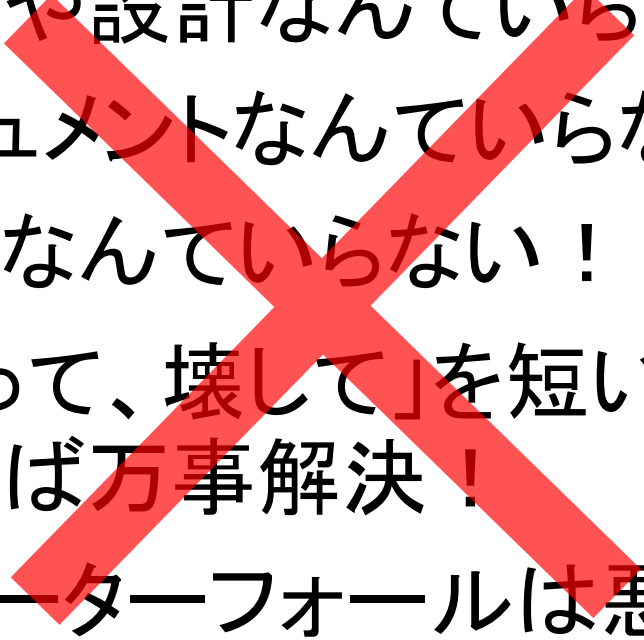
アジャイルが
「楽をする為の免罪符」
的に曲解されて扱われている
ケースがあるように思います

※もちろんちゃんと運用されている方もたくさんおられます

©SQUARE-ENIX 2011

よくあるアジャイルの誤解

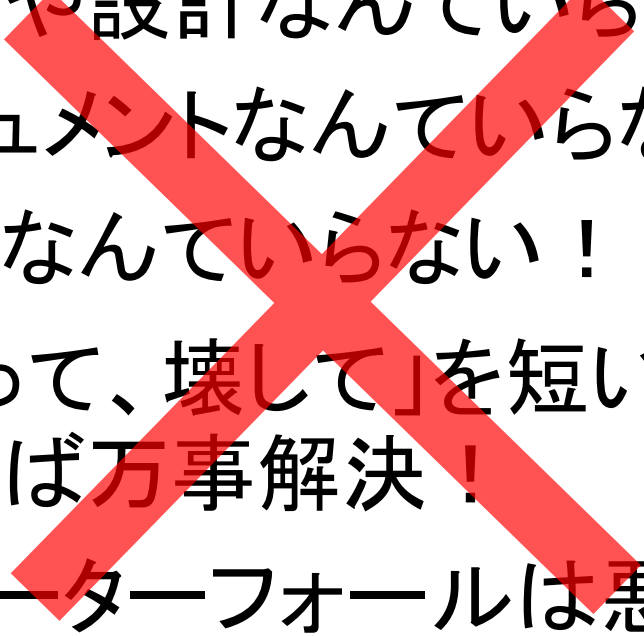
- 仕様や設計なんていらない！
- ドキュメントなんていらない！
- 計画なんていらない！
- 「作って、壊して」を短いイテレーションで繰り返せば万事解決！
- ウォーターフォールは悪！



©SQUARE-ENIX 2011

よくあるアジャイルの誤解

- 仕様や設計なんていらない！
- ドキュメントなんていらない！
- 計画なんていらない！
- 「作って、壊して」を短いイテレーションで繰り返せば万事解決！
- ウォーターフォールは悪！



これは幻想です

©SQUARE-ENIX 2011

よくあるアジャイルの誤解

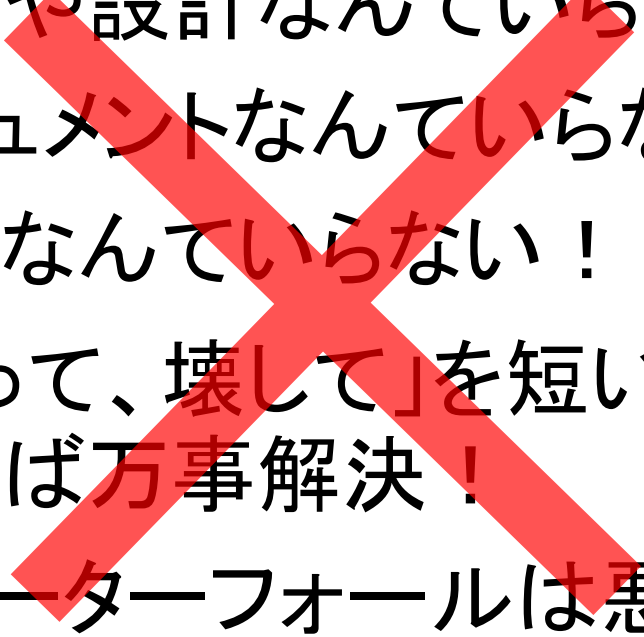
- 仕様や設計なんていらない！
- ドキュメントなんていらない！
- 計画なんていらない！
- 「作って、壊して」を短いイテレーションで繰り返せば万事解決！
- ウォーターフォールは悪！

繰り返します

©SQUARE-ENIX 2011

よくあるアジャイルの誤解

- 仕様や設計なんていらない！
- ドキュメントなんていらない！
- 計画なんていらない！
- 「作って、壊して」を短いイテレーションで繰り返せば万事解決！
- ウォーターフォールは悪！



これは幻想です

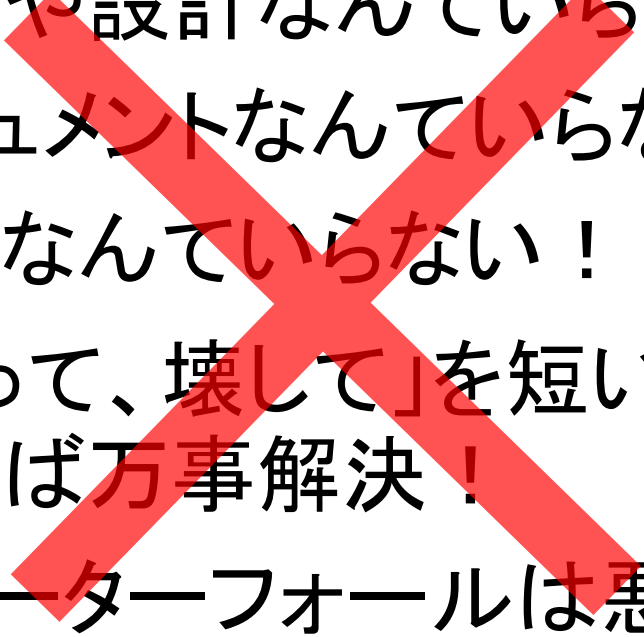
©SQUARE-ENIX 2011

アジャイルを誤解して
導入するとむしろ危険です

©SQUARE-ENIX 2011

よくあるアジャイルの誤解

- 仕様や設計なんていらない！
- ドキュメントなんていらない！
- 計画なんていらない！
- 「作って、壊して」を短いイテレーションで繰り返せば万事解決！
- ウォーターフォールは悪！



なぜ間違いなのか考察してみます

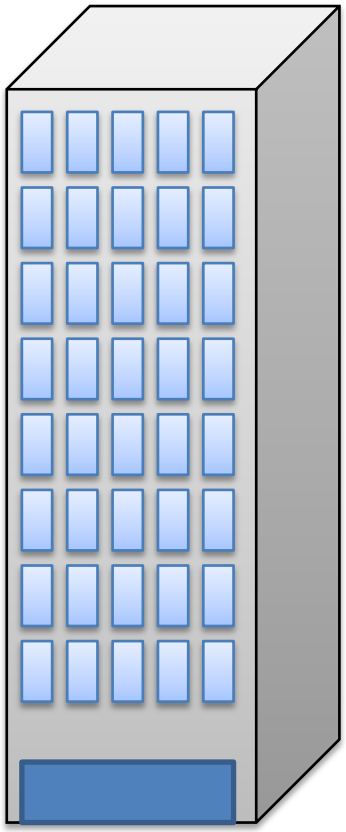
©SQUARE-ENIX 2011

「建物を建てるプロジェクト」を想像します

©SQUARE-ENIX 2011

建物を建てる

超高層ビル

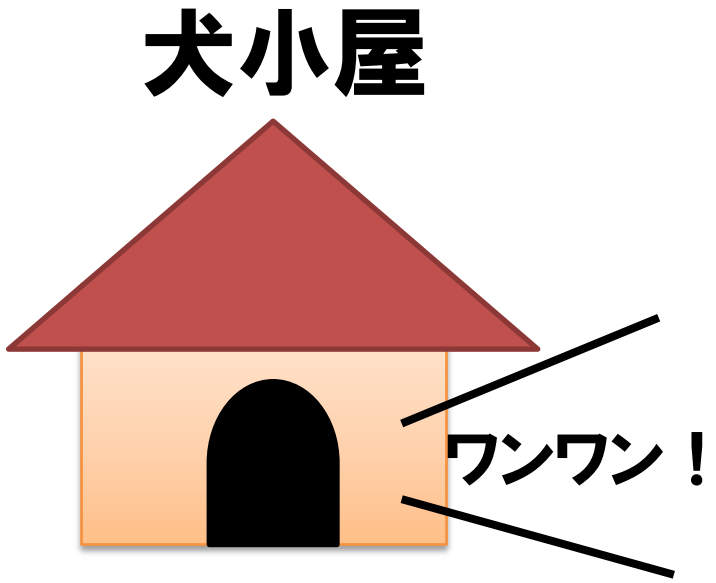


犬小屋

©SQUARE-ENIX 2011

犬小屋を建てる場合

- 一人でもできる
- せいぜい2、3日もあれば建てられる
- すべての作業を容易に洗い出す事ができる
- 一つ一つの作業の時間を予想しやすい
- 最悪やり直してもどうにかなる
- 計画ゼロでもどうにかなる

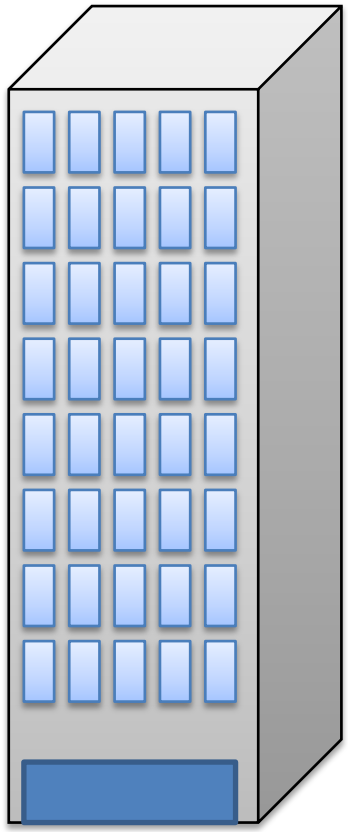


©SQUARE-ENIX 2011

超高層ビルを建てる場合

- 途方もない人数が必要
- 構想、設計から始めて数年かかる
- すべての作業を見通すことは大変難しい
- 一つ一つの作業の時間を予想しにくい
- やり直しがきかない
  - 例）40階建てのつもりだったけどやっぱり60階にしよう（無理！）

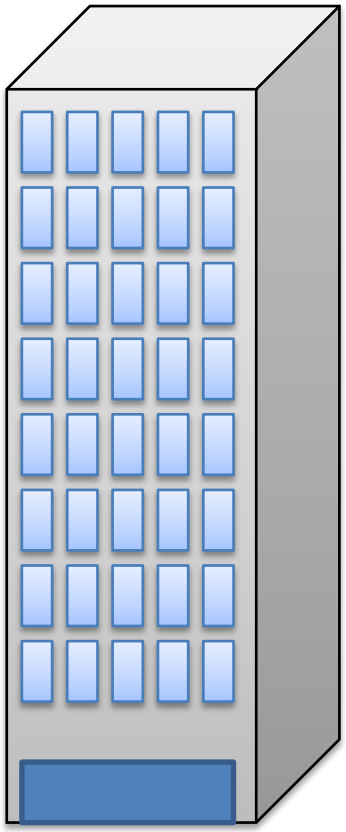
超高層ビル



©SQUARE-ENIX 2011

建物を建てる

超高層ビル

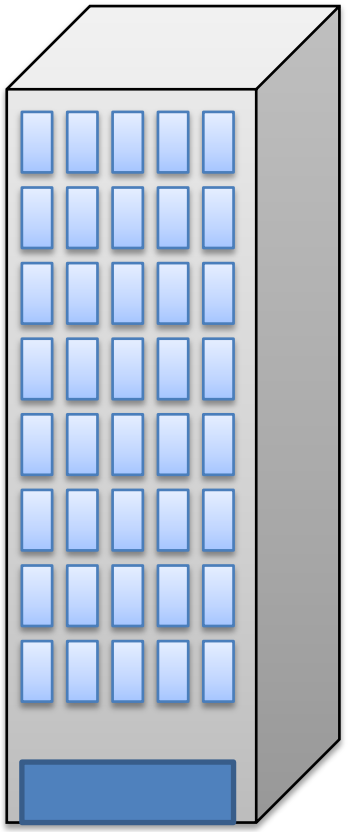


犬小屋

©SQUARE-ENIX 2011

建物を建てる

超高層ビル

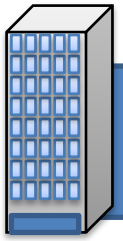


同じ方法論が
通用する？

犬小屋

©SQUARE-ENIX 2011

犬小屋



超高層ビル

犬小屋	超高層ビル
一人でもできる	途方もない人数が必要
せいぜい2、3日もあれば建てられる	構想、設計から始めて数年かかる
すべての作業を容易に洗い出すことができる	すべての作業を見通すことは大変難しい
一つ一つの作業の時間を予想しやすい	一つ一つの作業の時間を予想しにくい
最悪やり直してもどうにかなる	やり直しがきかない

©SQUARE-ENIX 2011

よくあるアジャイルの誤解

- 仕様や設計なんていらない！
- ドキュメントなんていらない！
- 計画なんていらない！
- 「作って、壊して」を短いイテレーションで繰り返せば万事解決！
- ウォーターフォールは悪

犬小屋を建てるならば確かにそれもアリかも・・・

©SQUARE-ENIX 2011

よくあるアジャイルの誤解

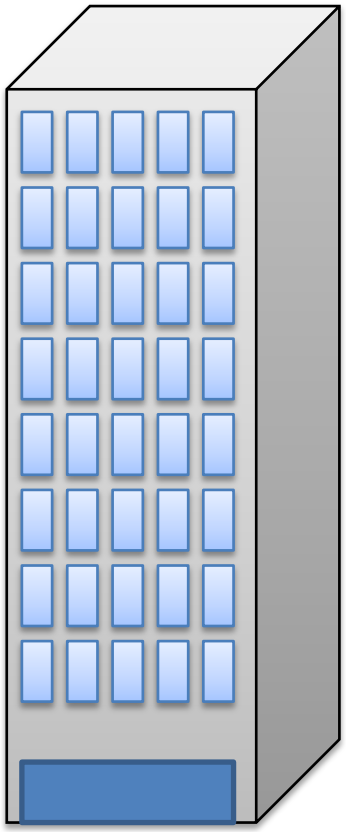
- 仕様や設計なんていらない！
- ドキュメントなんていらない！
- 計画なんていらない！
- 「作って、壊して」を短いイテレーションで繰り返せば万事解決！
- ウォーターフォールは悪

でも高層ビル建築でこういうポリシーでは大破綻

©SQUARE-ENIX 2011

建物を建てる

超高層ビル



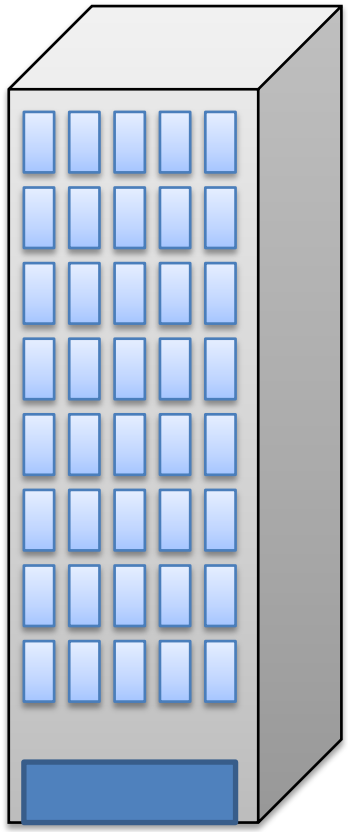
犬小屋

©SQUARE-ENIX 2011

建物を建てる

超高層ビル

同じ方法論は
通用しない！



犬小屋

©SQUARE-ENIX 2011

建物など
“規模や問題が目に見えやすい物”
で考えれば誰にでも簡単に
イメージ出来る話でしたね

©SQUARE-ENIX 2011

超高層ビルを建てる場合に
調査も設計も計画もなにも
せずにいきなり柱を立て始める
人はひとりもいませんよね？

©SQUARE-ENIX 2011

地盤調査や立地調査をし、資金計画やビジネス計画も行い、外観設計、内装設計、強度計算、素材実験、コスト計算、リスク見通し、人員計画、作業洗い出し、工数見積もり、なんでもやれる限りの事前準備をしないととてもまともな超高層ビルが建つ気がしないですよね？

©SQUARE-ENIX 2011

でもなぜかソフトウェア開発になると人は間違えてしまいます

©SQUARE-ENIX 2011

調査しない・・・
戦略を立てない・・・
設計しない・・・
計画しない・・・

©SQUARE-ENIX 2011

それでも中型・大型プロジェクトを
スタートさせてしまう無謀な
事例は決して少なくないと考えます

※きちんとやられている方の事は指しておりません

©SQUARE-ENIX 2011

“規模や問題が目に見えにくい”
ので自分が大破綻に向かってまっしぐらなことを全く自覚せずに、問題が表面化してどうにもならなくなるまで作業を進めることができてしまうのがソフトウェア開発の恐ろしい所なのです

©SQUARE-ENIX 2011

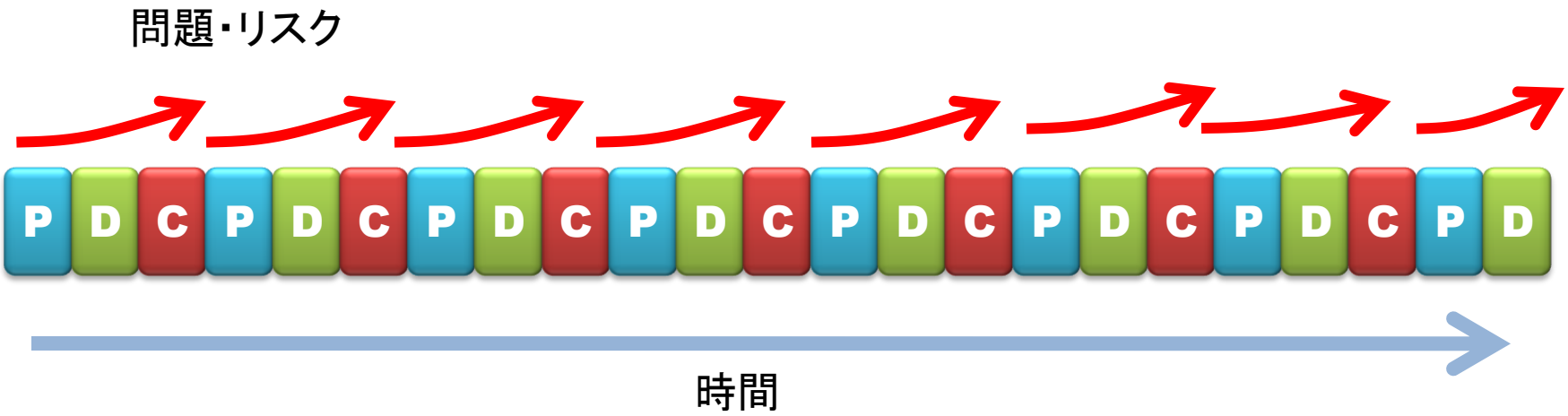
話は戻りまして
さっきのおさらい

©SQUARE-ENIX 2011

Plan-Do-Checkサイクル

©SQUARE-ENIX 2011

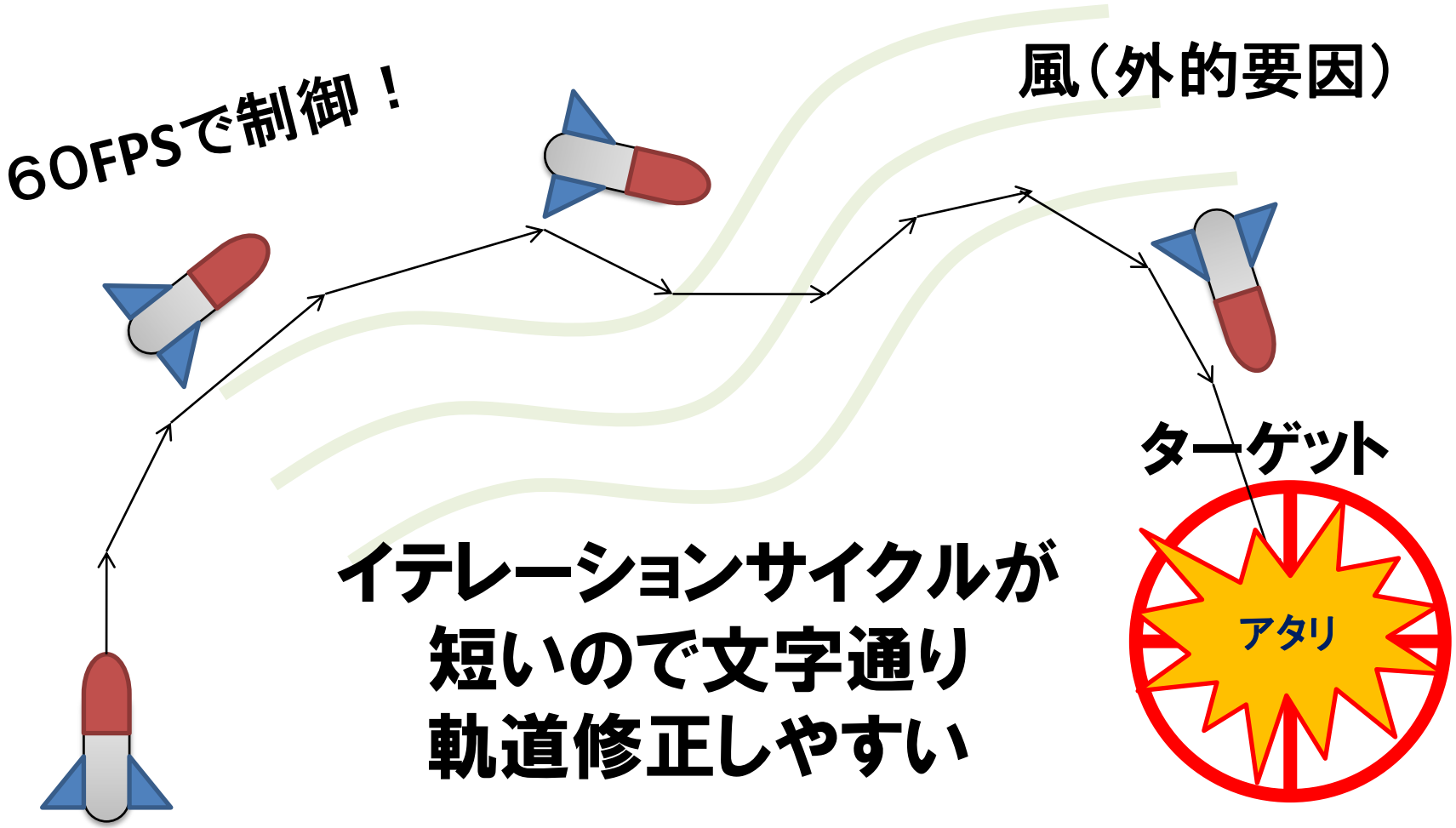
良いプロジェクト



短いサイクルで
計画⇒実行⇒振り返り⇒再計画⇒実行⇒振り返り⇒再計画・・・
と回すことで問題やリスクが早い段階で摘み取られるので、安定したプロジェクト運営が行われる。

©SQUARE-ENIX 2011

ミサイルを飛ばす（良い例）



©SQUARE-ENIX 2011

イテレーションサイクルさえ細かく回してれば万事解決的に見えますが・・・

©SQUARE-ENIX 2011

本当にそうでしょうか

©SQUARE-ENIX 2011

確かにイテレーションを
細かく回すことは
絶対的に重要です
疑う余地がありません

©SQUARE-ENIX 2011

ですがそれだけでは
越えられない壁があります

©SQUARE-ENIX 2011

再び考察

©SQUARE-ENIX 2011

「ダンジョンを歩いて抜ける」プロジェクト

©SQUARE-ENIX 2011

スタートとゴール

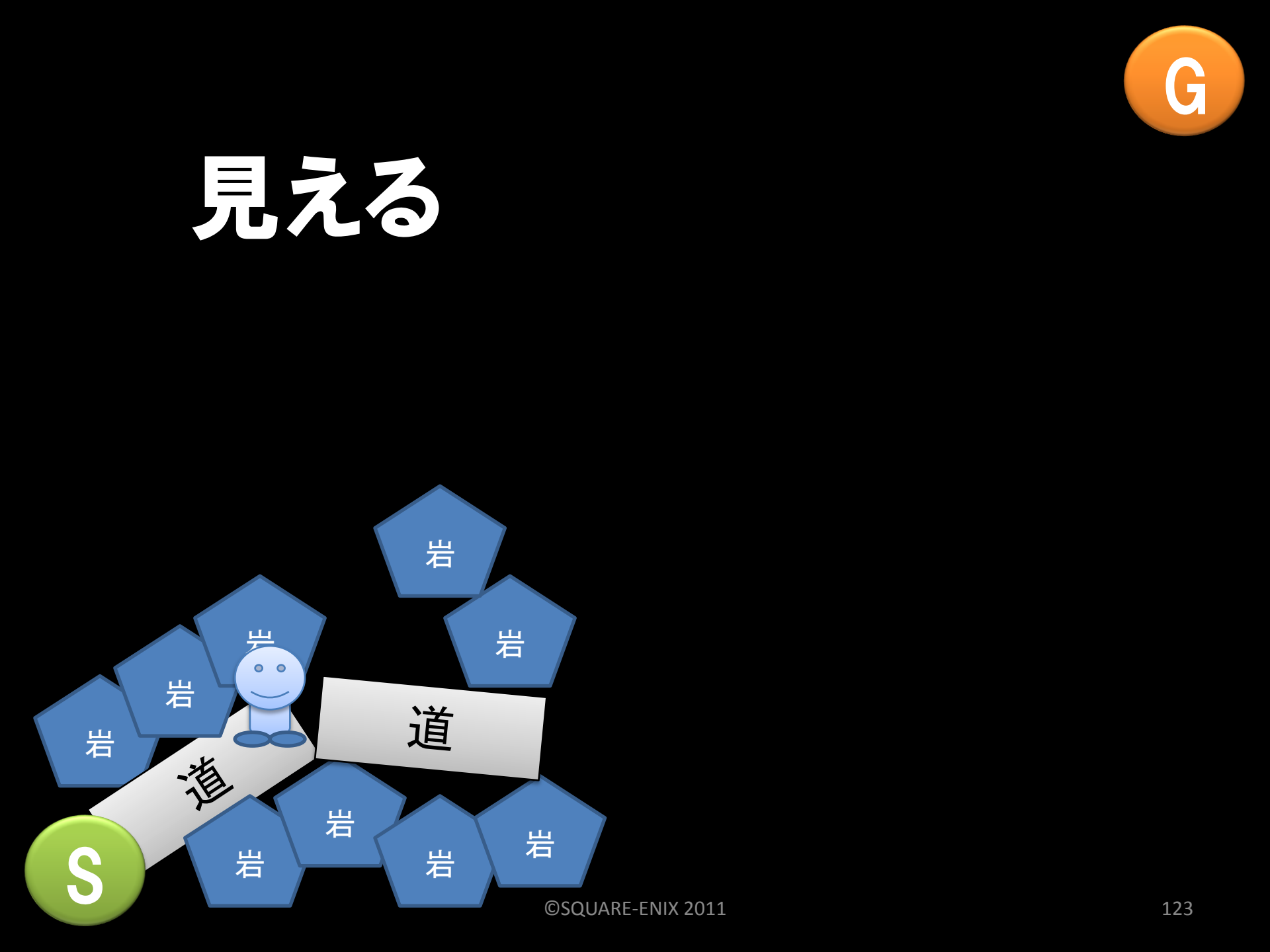
©SQUARE-ENIX 2011

©SQUARE-ENIX 2011

G

歩く

©SQUARE-ENIX 2011



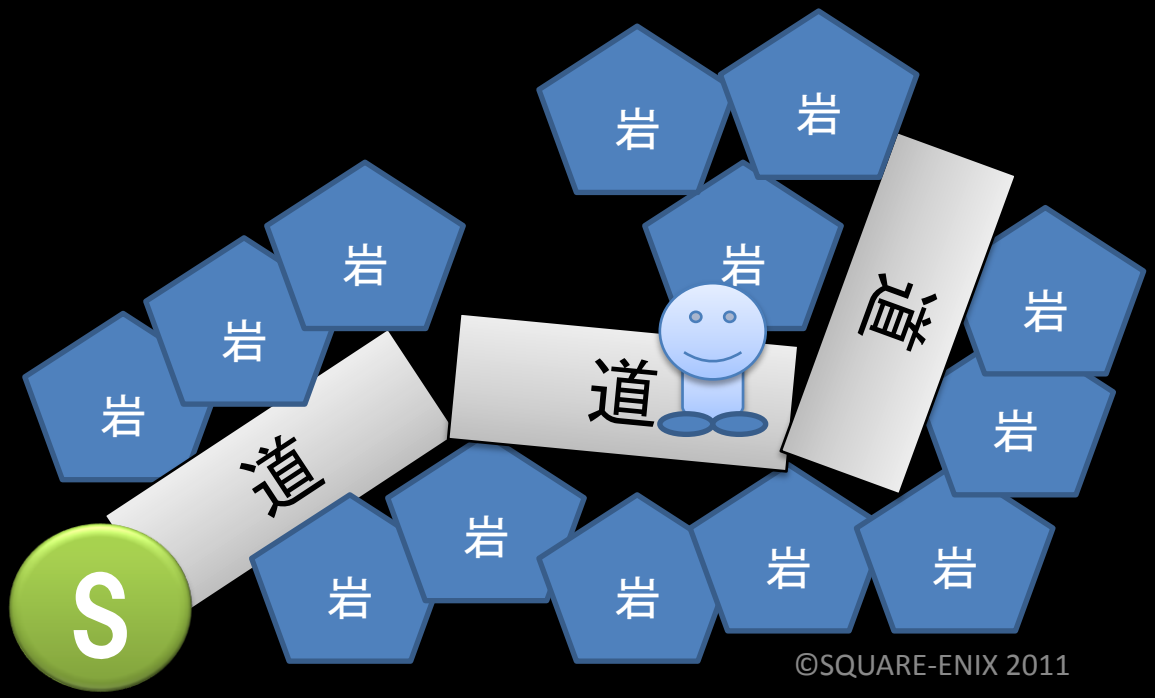
©SQUARE-ENIX 2011

歩く

©SQUARE-ENIX 2011

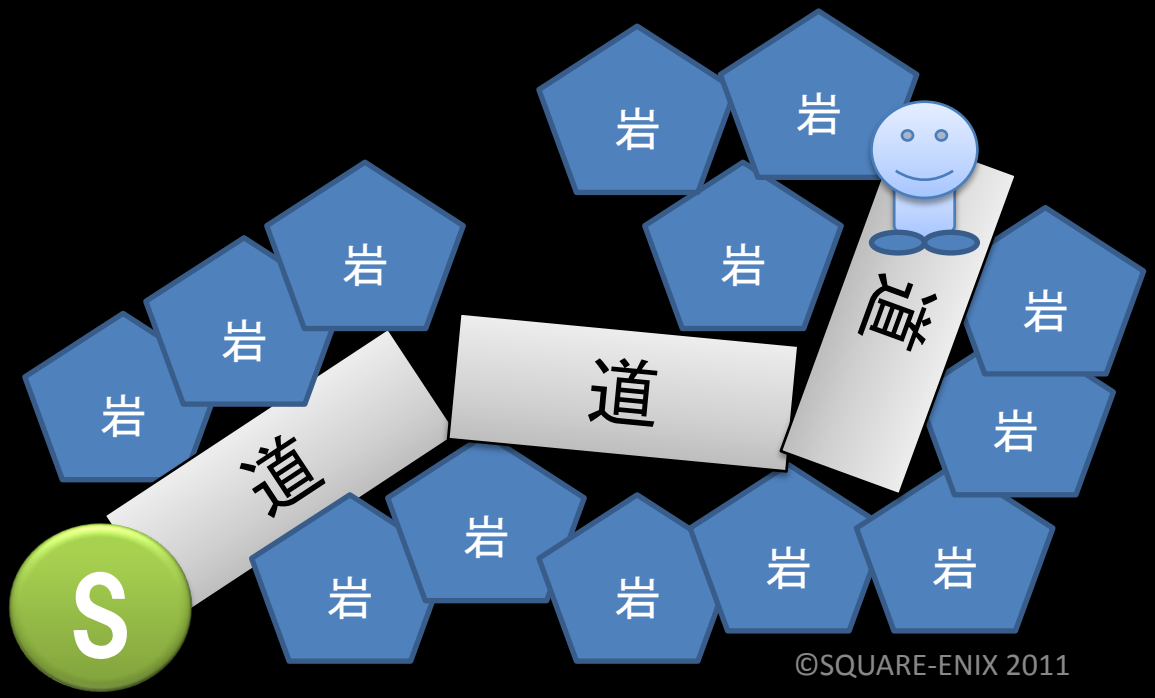
G

見える



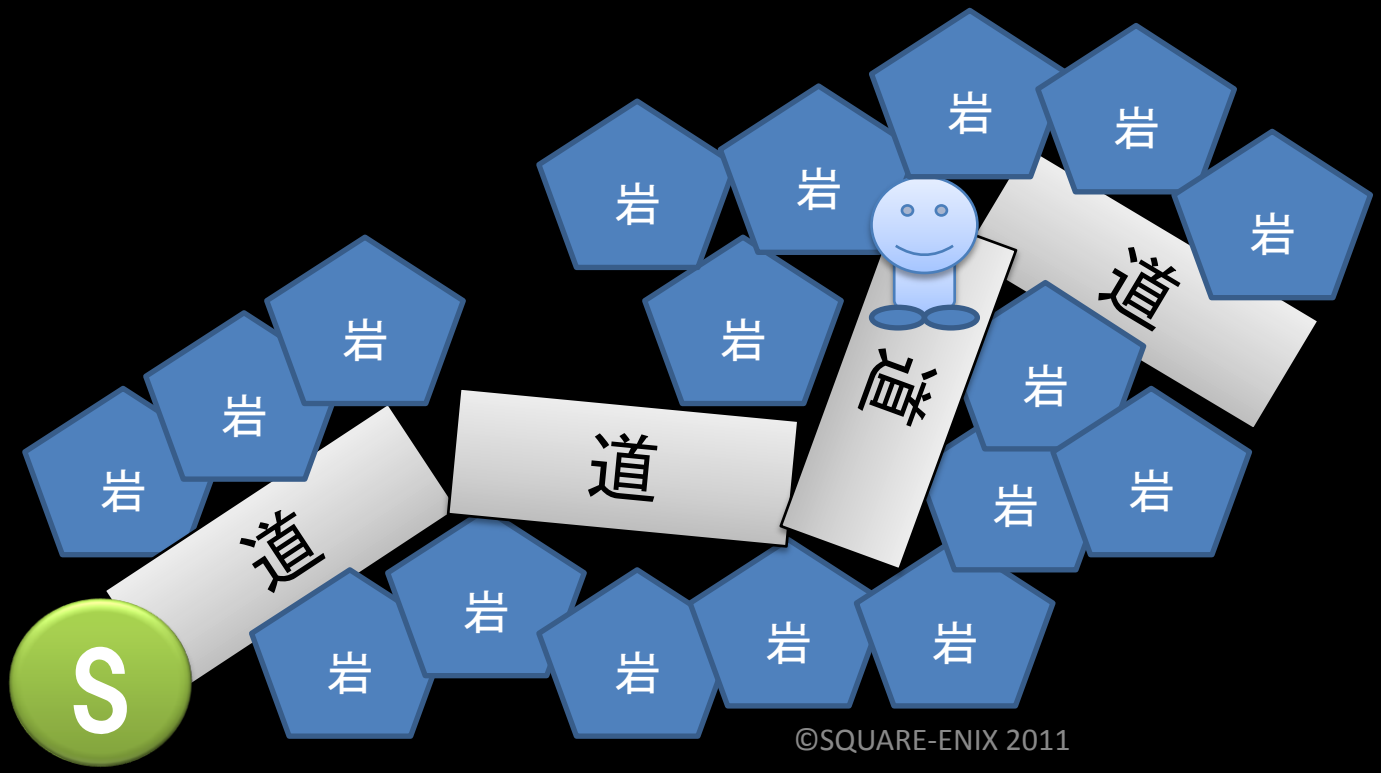
©SQUARE-ENIX 2011

歩く



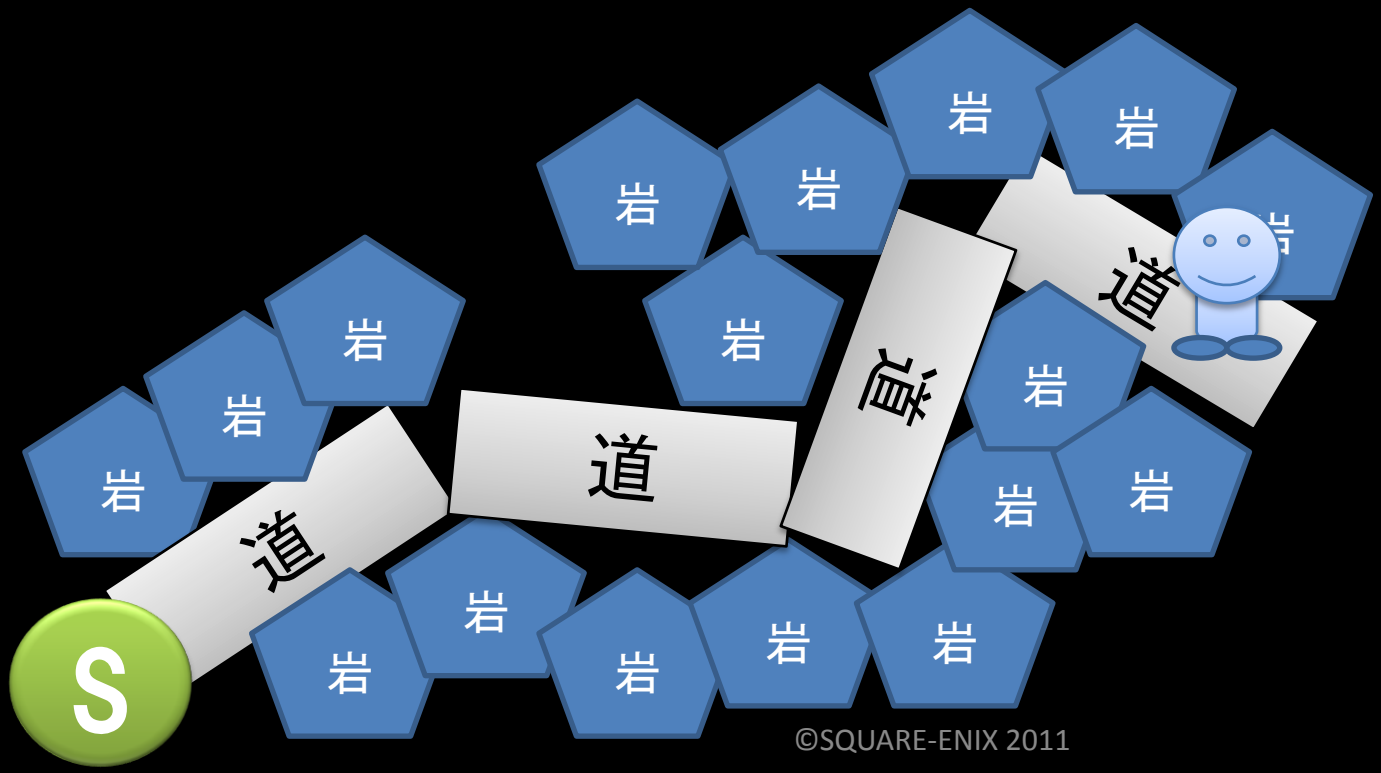
©SQUARE-ENIX 2011

見える



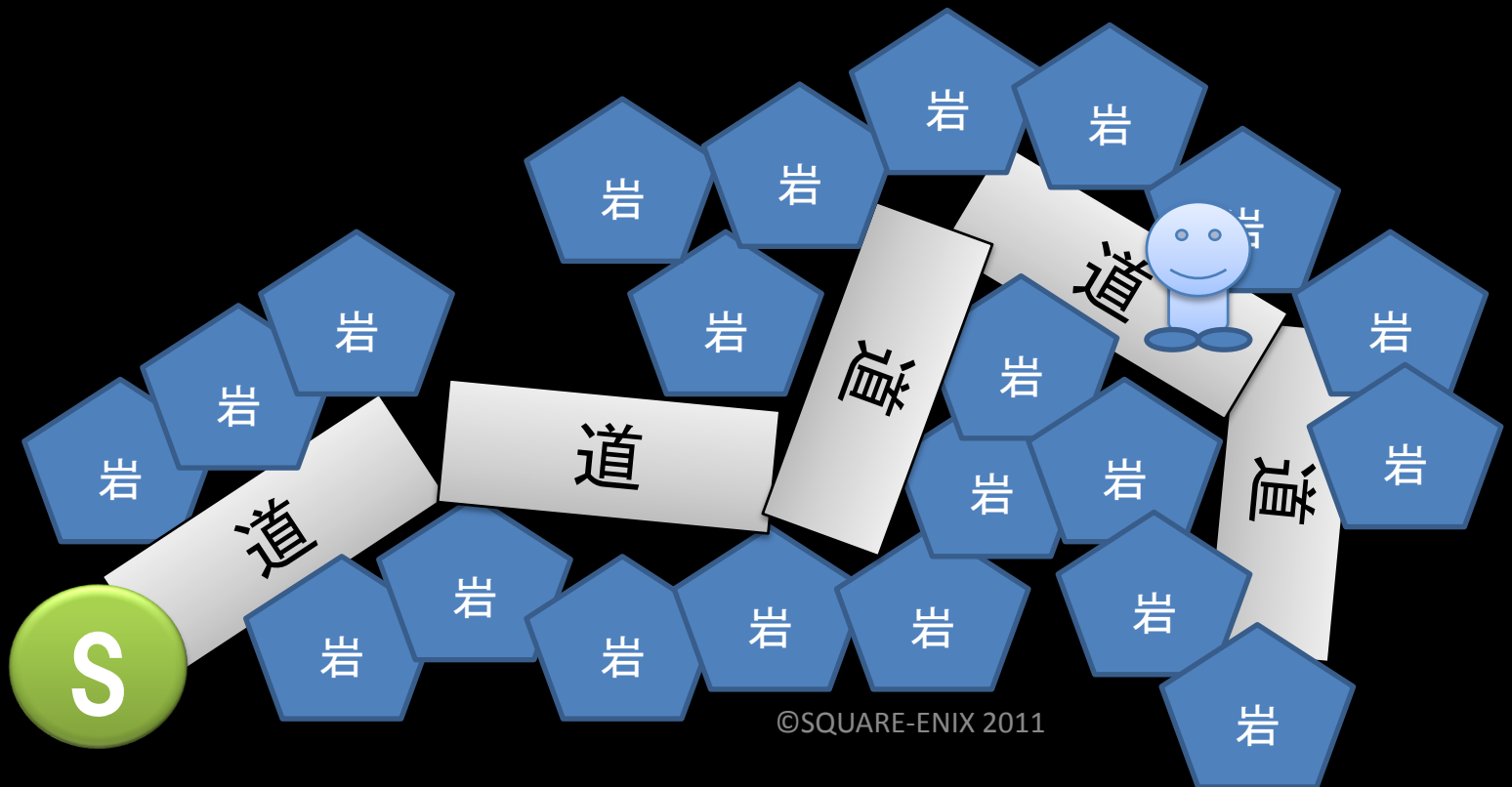
©SQUARE-ENIX 2011

歩く



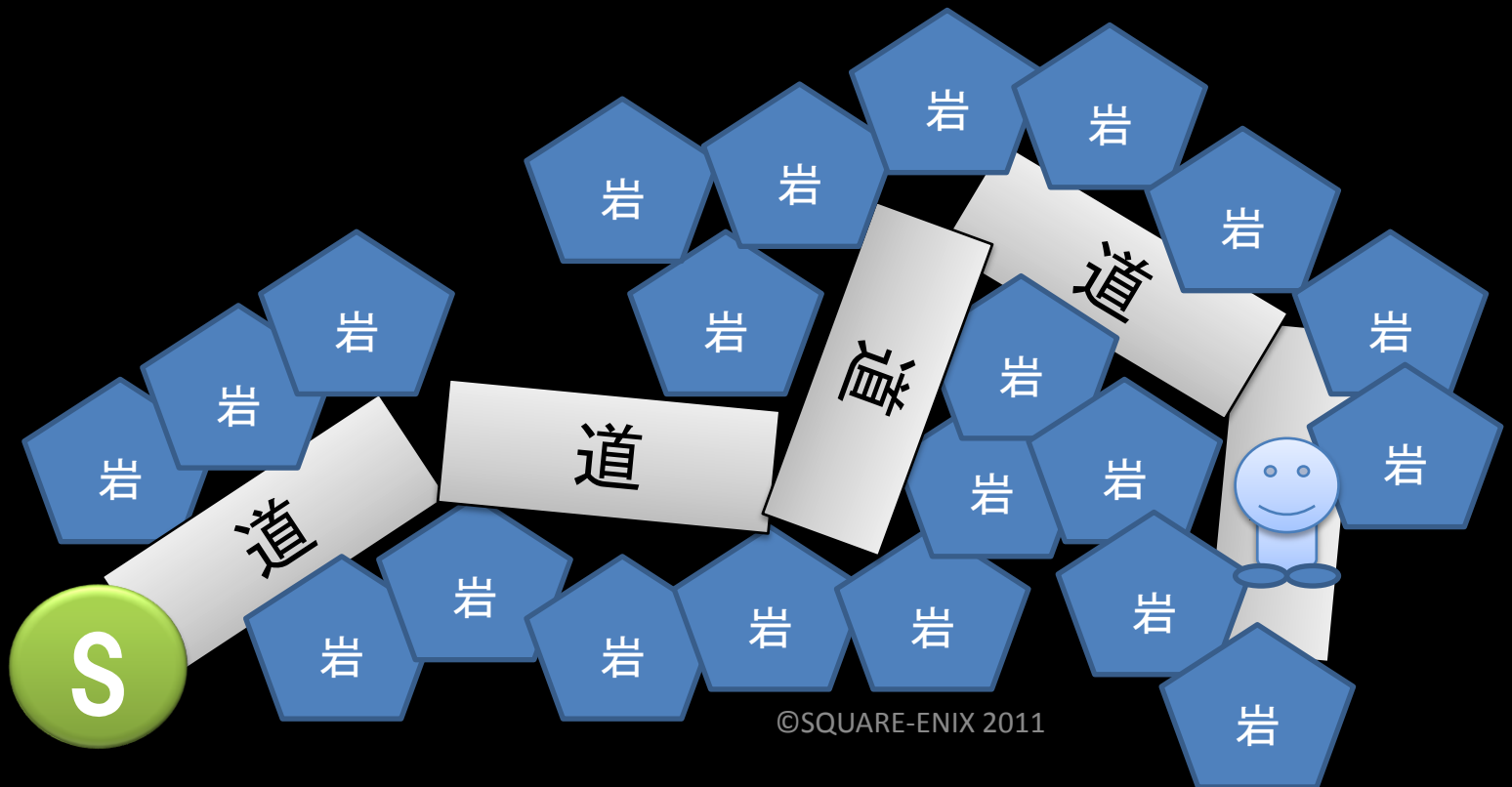
©SQUARE-ENIX 2011

見える



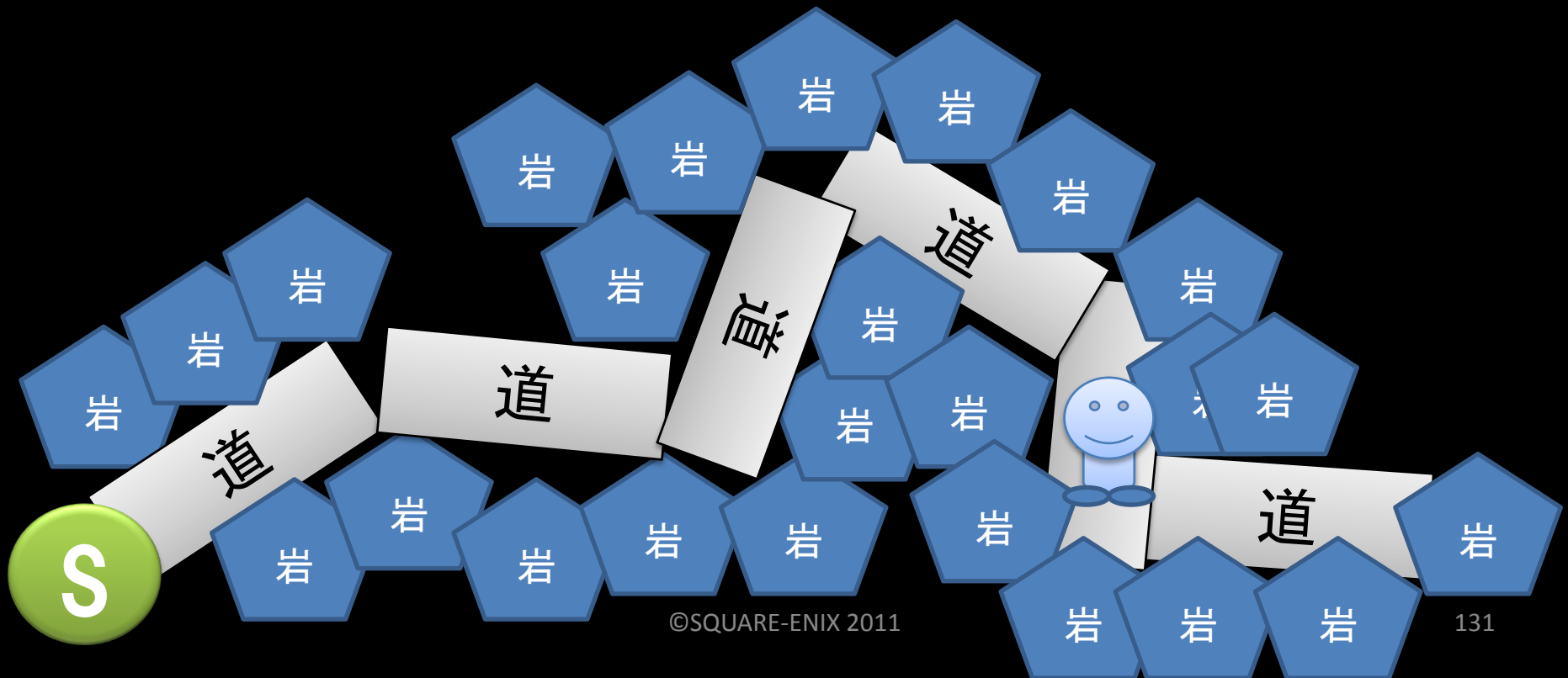
©SQUARE-ENIX 2011

歩く



©SQUARE-ENIX 2011

見える

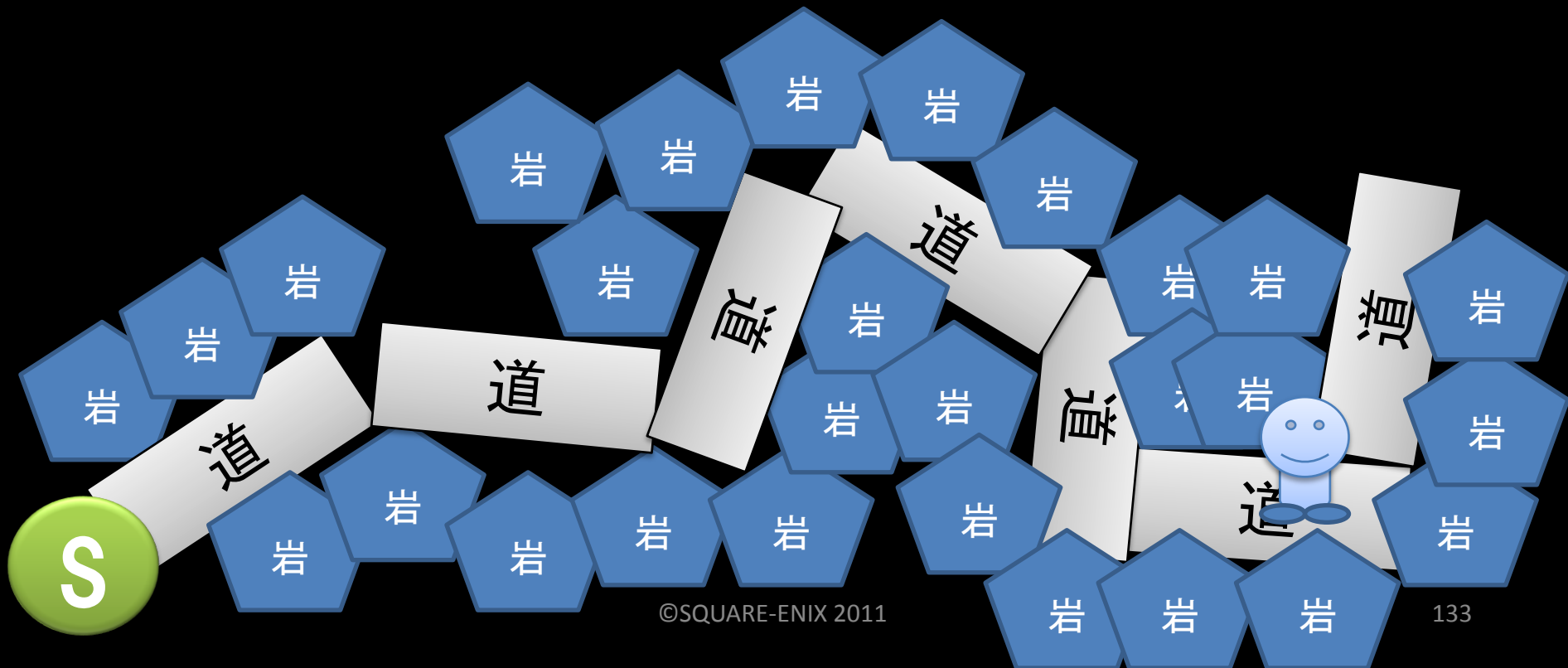


©SQUARE-ENIX 2011

歩く

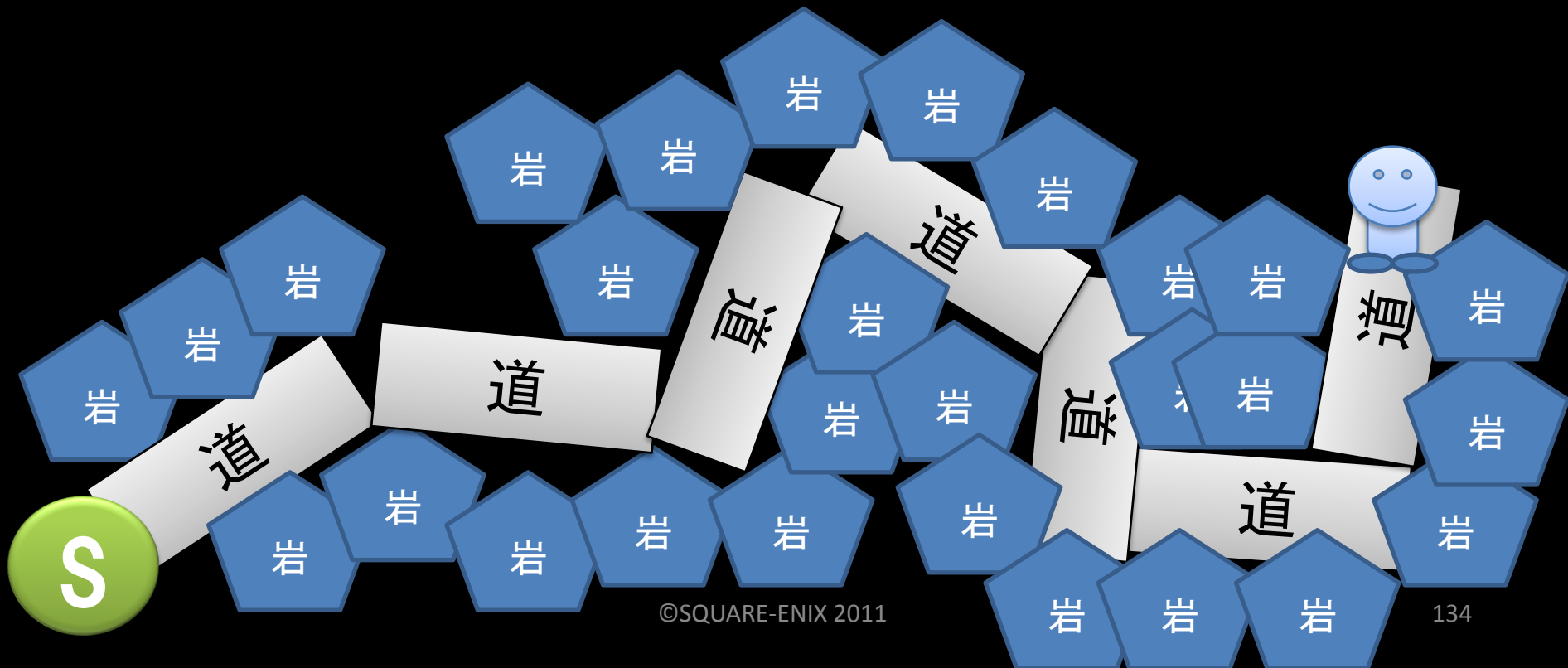
©SQUARE-ENIX 2011

順調順調



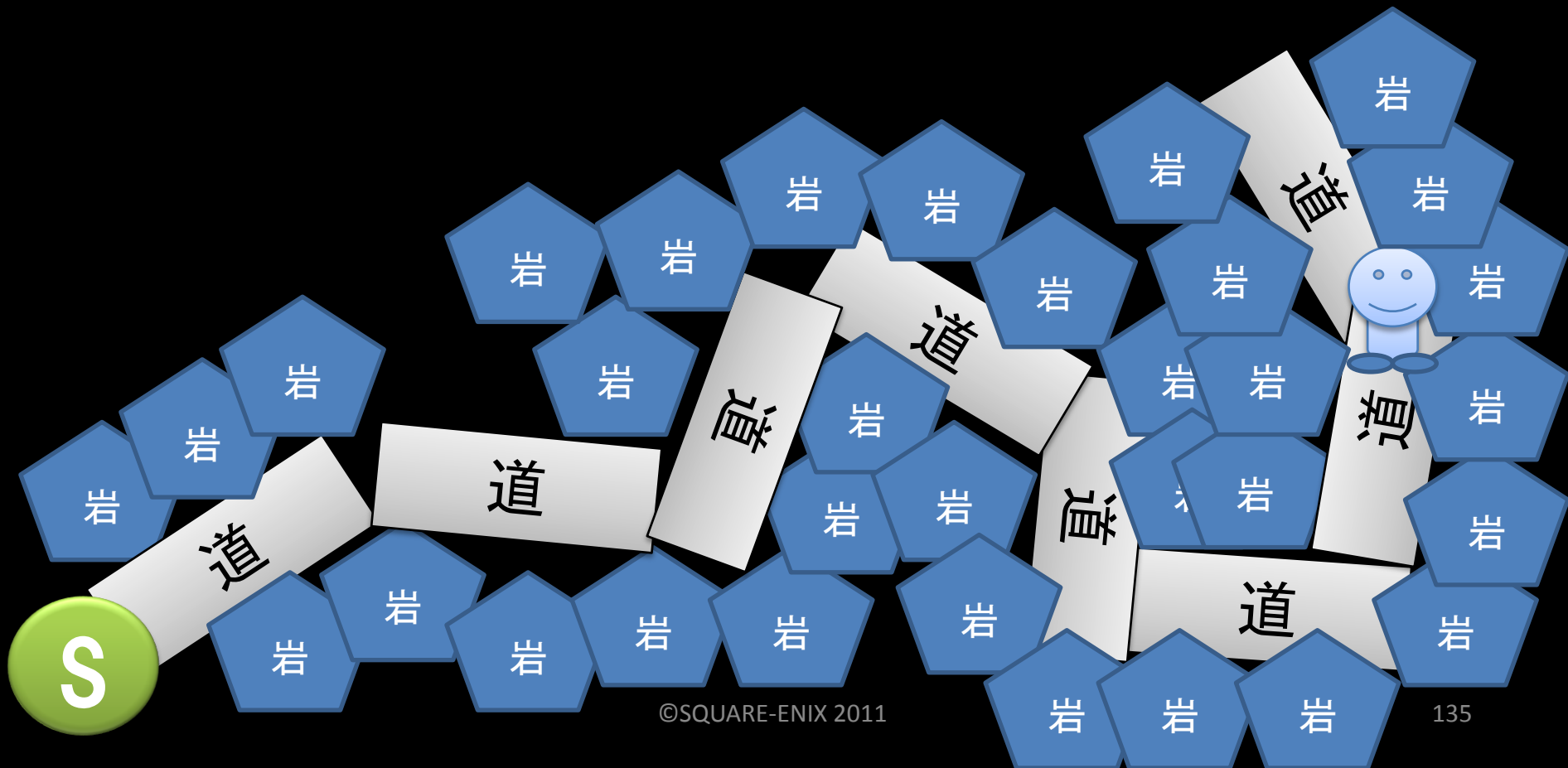
©SQUARE-ENIX 2011

イテレーション万歳！



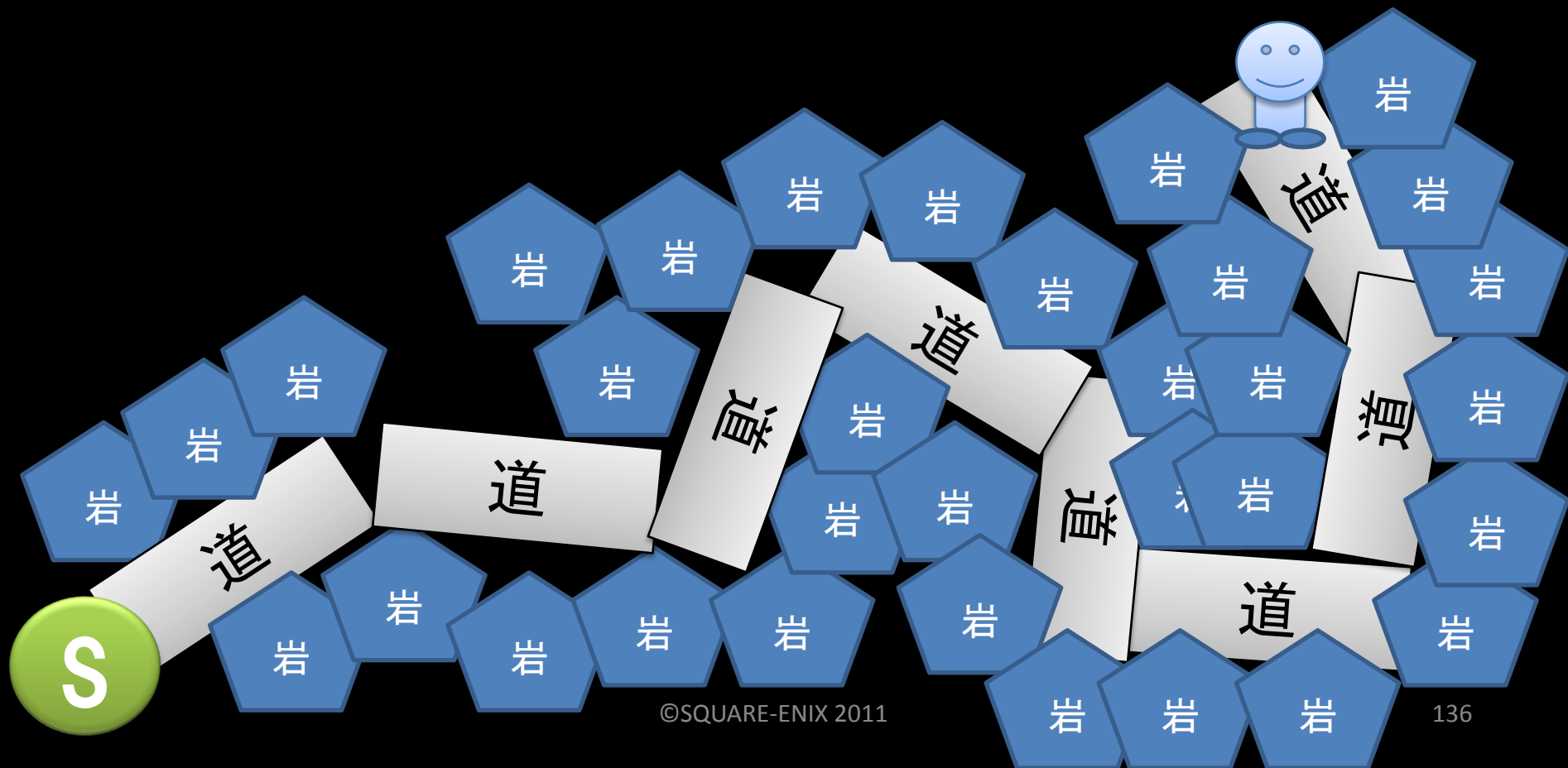
©SQUARE-ENIX 2011

オレって天才？



©SQUARE-ENIX 2011

楽勝だね♪



©SQUARE-ENIX 2011

もうすぐゴールだ！

©SQUARE-ENIX 2011

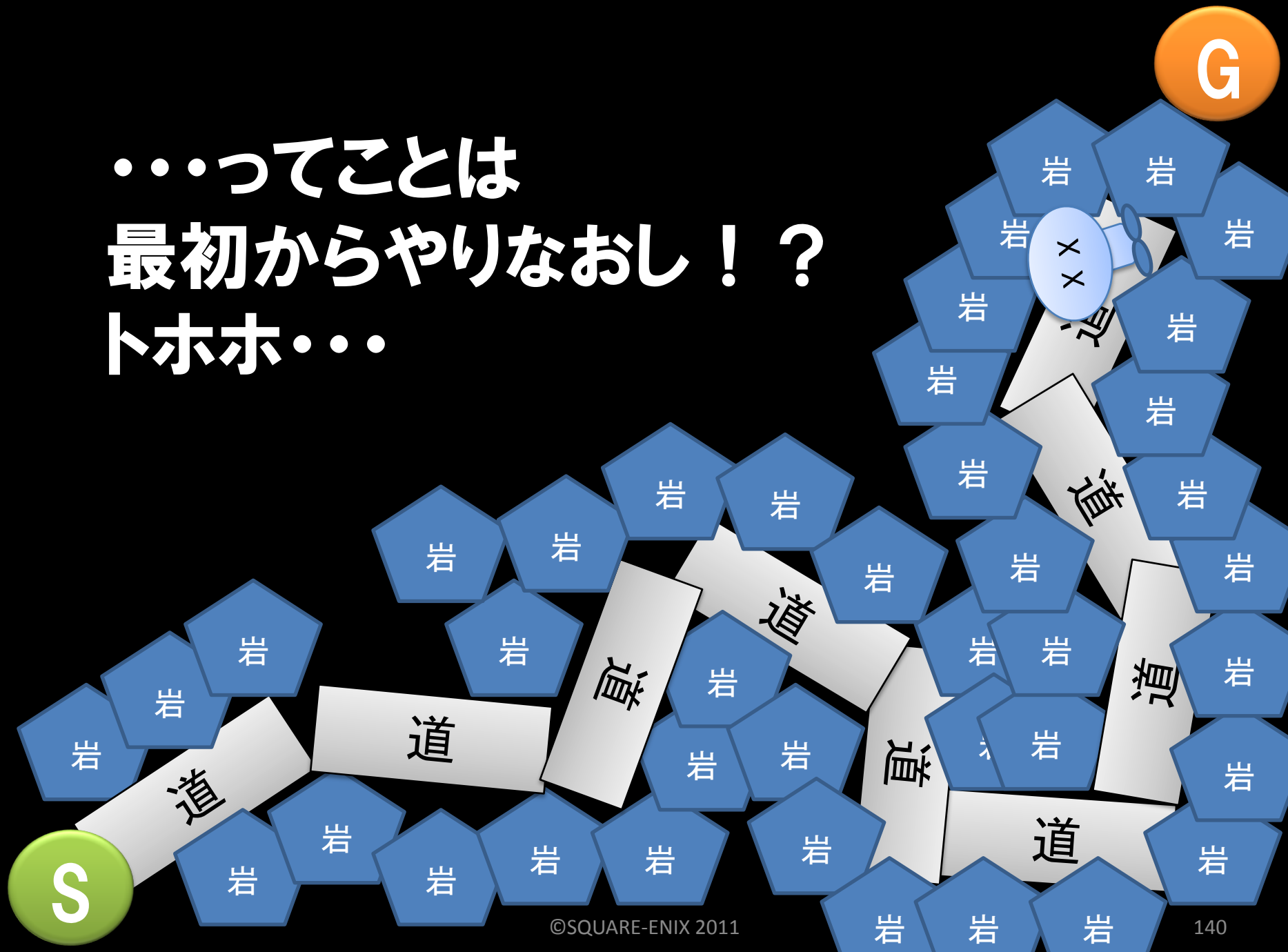
ゴンッ！！

©SQUARE-ENIX 2011

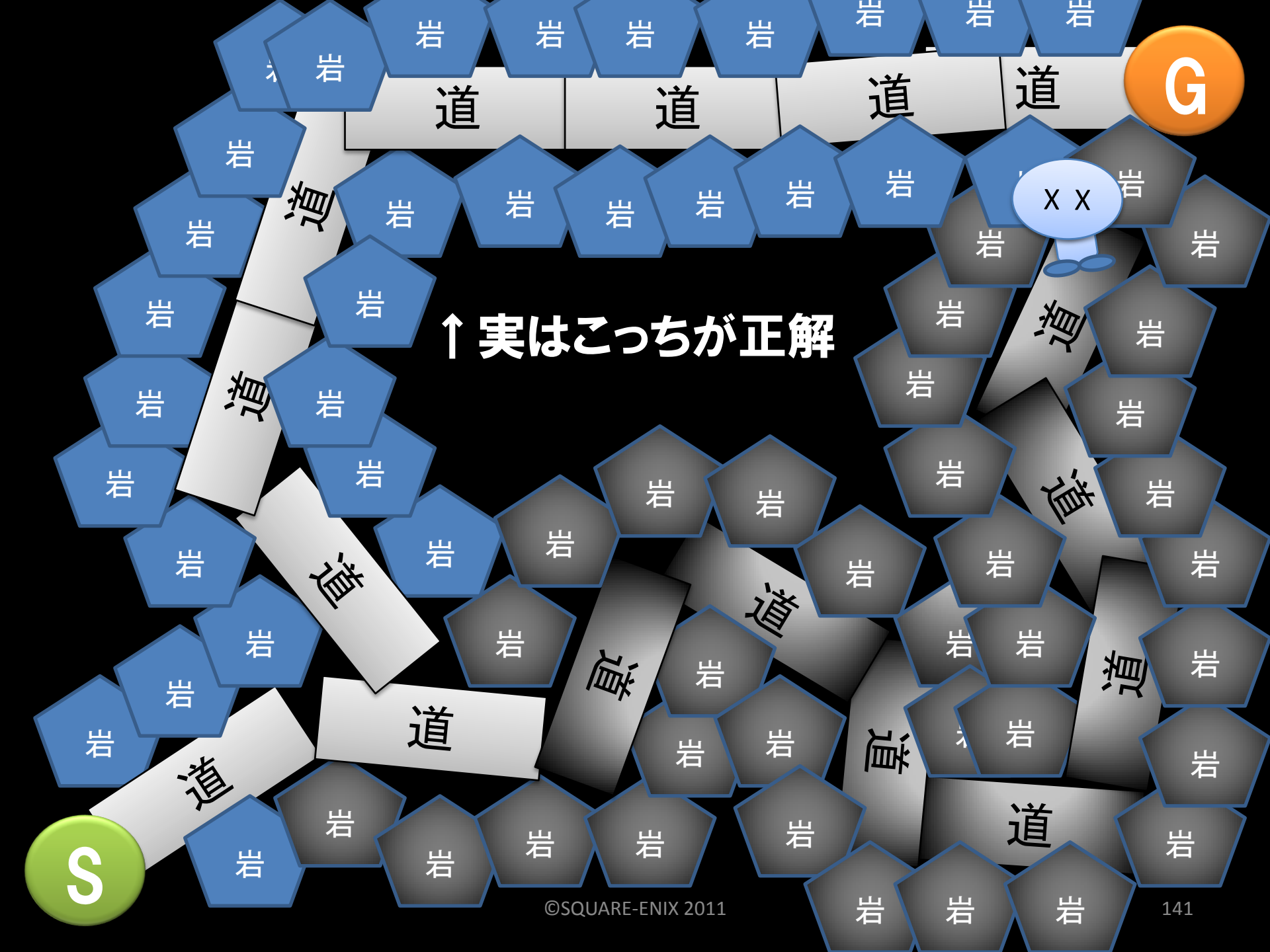
うっそ～！
行き止まり！！

©SQUARE-ENIX 2011

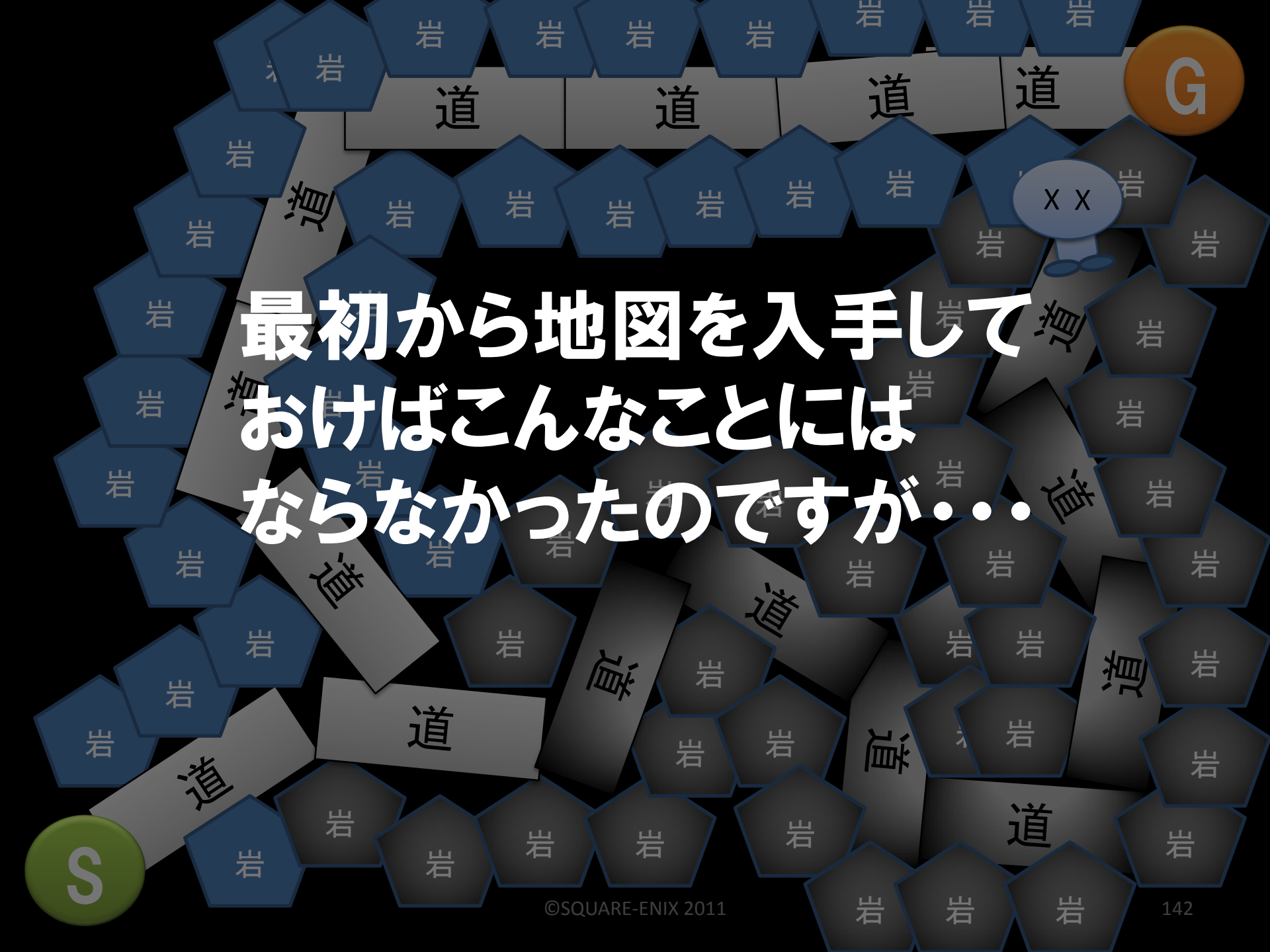
・・・ってことは
最初からやりなおし！？
トホホ・・・



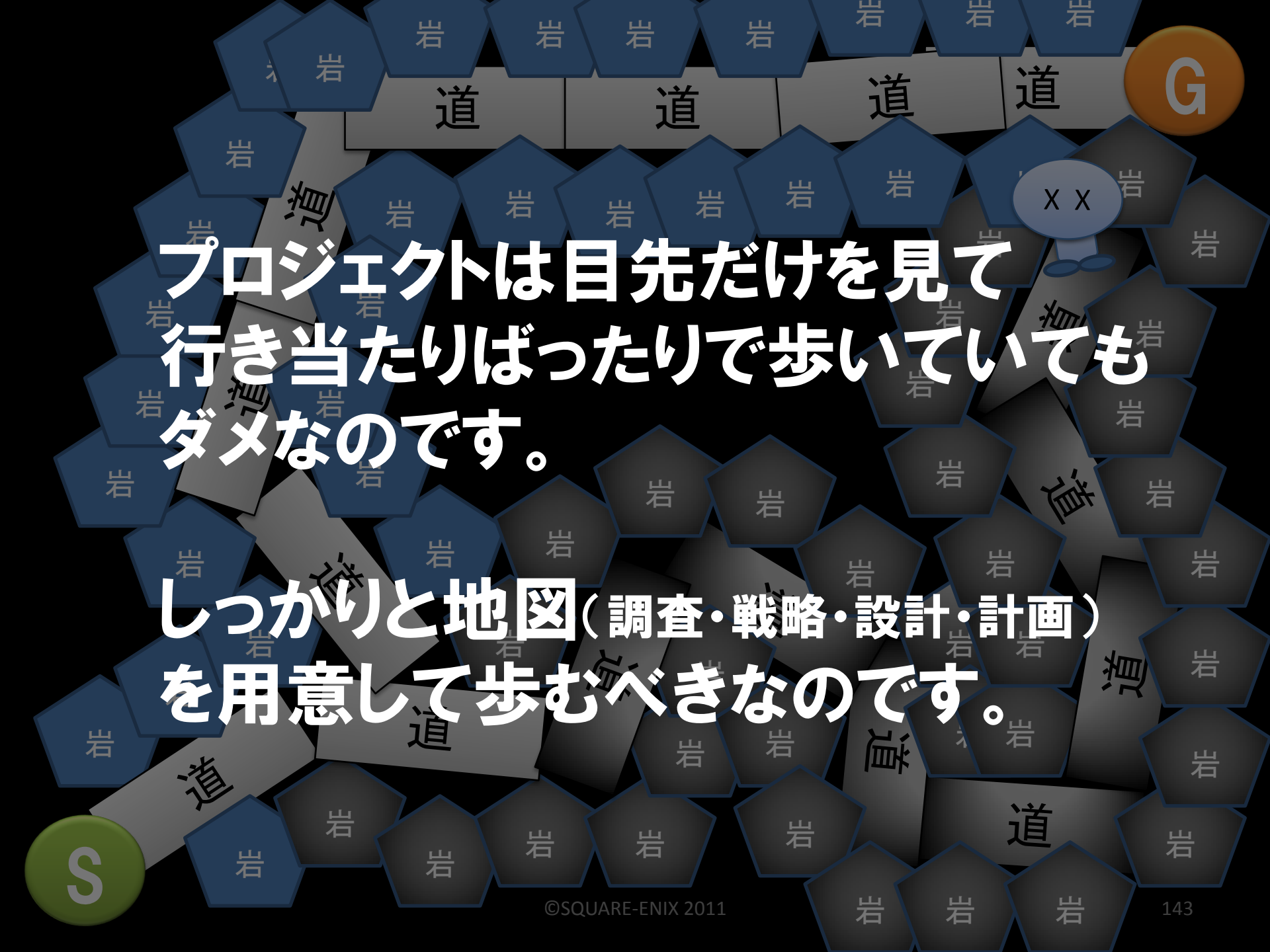
©SQUARE-ENIX 2011



©SQUARE-ENIX 2011



©SQUARE-ENIX 2011



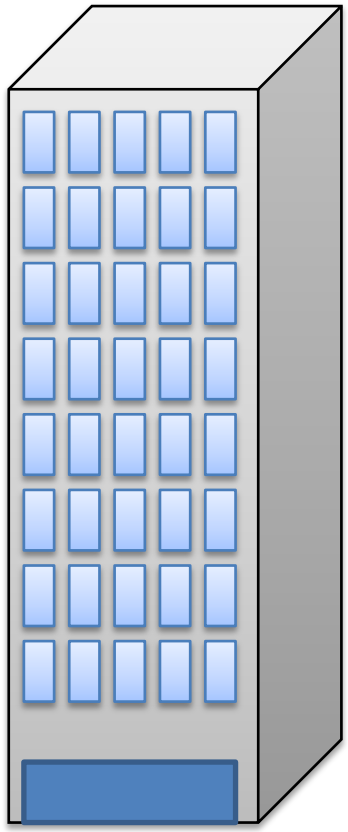
プロジェクトは目先だけを見て
行き当たりばったりで歩いていても
ダメなのです。

しっかりと地図（調査・戦略・設計・計画）
を用意して歩むべきなのです。

©SQUARE-ENIX 2011

こんな例や

超高層ビル



©SQUARE-ENIX 2011

こんな例からも

©SQUARE-ENIX 2011

調査・戦略・設計・計画の重要性が示されています

©SQUARE-ENIX 2011

ソフトウェア開発だけが
特別なのでしょうか？

本当に調査・戦略・設計・計画を
軽視してソフトウェア開発は
できるのでしょうか？

©SQUARE-ENIX 2011

調査・戦略・設計・計画なしで
一定サイズ以上の
ソフトウェアを作ることは
極めて危険です

©SQUARE-ENIX 2011

調査・戦略・設計・計画から逃げてはいけないのです！

©SQUARE-ENIX 2011

当たり前の準備を
地道にやる

©SQUARE-ENIX 2011

それが不確実性を
乗りこなす秘訣です

©SQUARE-ENIX 2011

不確実性を乗りこなす

- **事前対策**（調査・戦略・設計・計画）
  - **不確実性を減らす**
    - 見通しが悪い部分の調査や検証/実験をしっかりと行う
    - 作業データを取り、フィードバックをかけて予測精度を向上させる
    - 計画規模を小さくする
      - どうせ計画は膨らむのだから最初に目いっぱいコンパクトにする
    - 設計をなるべく詳細に行う
    - リスクの高い要素を予め極力減らす
  - **不確実でも良い事にする**
    - リスクの高い要素を無くしてても成立する設計にする
      - その要素が無くなっても商品の価値が下がらない設計にする
        - » **トカゲのしっぽ**としていつでも切れるようにする
- **事後対策**（イテレーション）
  - **変化や問題発生にすばやく気づく**
  - **変化や問題にすばやく対応する**

©SQUARE-ENIX 2011

いきなり歩くのは危険です

©SQUARE-ENIX 2011

確かに
慎重に歩けば（イテレーション）
足元の石ころで転ぶことは
避けられるかもしれません

©SQUARE-ENIX 2011

しかし、目先だけを見ていては
行く手に行き止まりや落とし穴
やモンスターや山や渓谷が
存在することに直前まで
なかなか気づけません

©SQUARE-ENIX 2011

足元(ショートレンジ)も
遠くの行く手(ロングレンジ)も
ちゃんと見て歩かないとゴールには
無事たどり着けないのです

©SQUARE-ENIX 2011

だから

地図（設計）や双眼鏡（計画）

を予め用意して、不測の事態を事前に
避けたり、もし事前に避けられなくても
それに気づき易く、対策し易くする
事前準備をしましょう

©SQUARE-ENIX 2011

地図や双眼鏡を作ってから歩き出しましょう

©SQUARE-ENIX 2011

そして
その地図や双眼鏡は
常にメインテナンスしましょう

※地図や双眼鏡のメインテナンスは大きなPDCサイクルと言えます

©SQUARE-ENIX 2011

不確実性を乗りこなす

- **事前対策**（調査・戦略・設計・計画）
  - **不確実性を減らす**
    - 見通しが悪い部分の調査や検証/実験をしっかりと行う
    - 作業データを取り、フィードバックをかけて予測精度を向上させる
    - 計画規模を小さくする
      - どうせ計画は膨らむのだから最初に目いっぱいコンパクトにする
    - 設計をなるべく詳細に行う
    - リスクの高い要素を予め極力減らす
  - **不確実でも良い事にする**
    - リスクの高い要素を無くしてても成立する設計にする
      - その要素が無くなっても商品の価値が下がらない設計にする
        - » **トカゲのしっぽ**としていつでも切れるようにする
- **事後対策**（イテレーション）
  - **変化や問題発生にすばやく気づく**
  - **変化や問題にすばやく対応する**

©SQUARE-ENIX 2011

事前対策も事後対策も
両方やることに意味があるのです
片方だけでは機能不全です

©SQUARE-ENIX 2011

極めてシンプルな話なのです

©SQUARE-ENIX 2011

しっかり準備、段取りして
しっかりイテレーションしましょう

©SQUARE-ENIX 2011

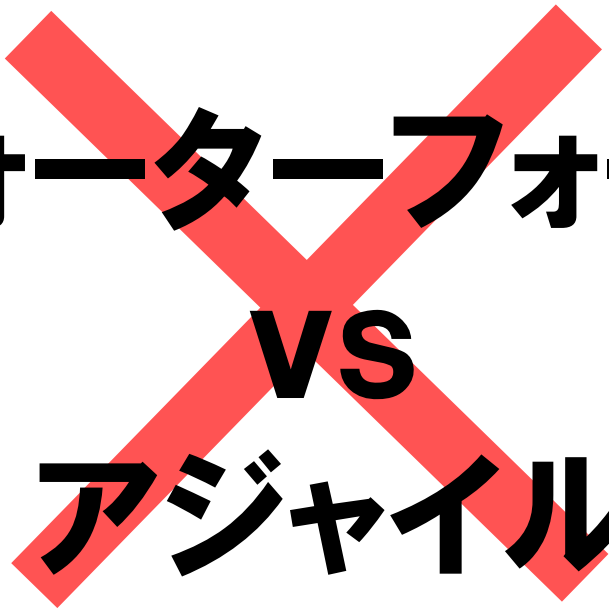
ウォーターフォール

VS

アジャイル

というような見方をついしてしまいがちですが

©SQUARE-ENIX 2011



対立構図は好ましくありません

©SQUARE-ENIX 2011

ウォーターフォール

×

アジャイル

©SQUARE-ENIX 2011

ウォーターフォール
×
アジャイル

それぞれの持ち味を生かしましょう

©SQUARE-ENIX 2011

ウォーターフォール派とかアジャイル派
といったように表面上の型やイメージや
定義にふりまわされるのはやめて、
プロジェクトマネジメントの本質を捉えて
ハイブリッド型の良いとこ取りの方法で
バランス良く行きましょう

©SQUARE-ENIX 2011

一旦まとめます

©SQUARE-ENIX 2011

一旦のまとめ

- **プロジェクトには当初に予想しきれないほど大きな不確実性が潜んでいる**
- **従って不確実性とうまく付き合う必要がある**
- **イテレーションを重視しよう**
  - こまめなPlan-Do-Checkサイクルで、不確実性から生まれる問題の早期発見と早期解決を繰り返そう
- **調査・戦略・設計・計画からは逃げない**
  - 不確実性を下げ、見通しを良くするための地図を準備しよう
  - それらはなるべくドキュメント化しよう

©SQUARE-ENIX 2011

さて、概念論はこれくらいにして
もう少し具体的な話に降りて行きます

©SQUARE-ENIX 2011

プロジェクトマネジメントの手順

©SQUARE-ENIX 2011

プロジェクトマネジメントの手順

1. **調査する**
2. **戦略を立てる**（リスク一覧/開発戦略マトリクス/価値空間）
3. **設計する**
4. **計画する**（中長期計画）
   - ① タスクを洗い出す（ユーザーストーリー/フィーチャー/タスク/ＬＯＤ式タスク分解）
   - ② 見積もる（２点見積もり/見積もりポーカー/ＬＯＤ式見積もり）
   - ③ 優先度を決める
   - ④ マイルストーンを定める
5. **スプリント**（４週間単位の制作イテレーション）
   - – 階層化ＰＤＣ
   - ① スプリント計画（成果物目標/スプリント計画会/タスク管理シート）
   - ② 日々の制作（朝会/タスク管理ボード）
   - ③ 週の振り返り（週報/週定例）
   - ④ スプリントの振り返り（スプリント報/スプリント振り返り会/成果物発表会）
   - ⑤ 対策・再計画
     - 戦略・設計の修正（ゲーム仕様、アート、プログラム設計などの修正）
     - 中長期計画の修正（タスクの追加・変更、見積もりの変更、優先度の変更）
     - リソースの修正（人員の追加や担当変更など）
   - ⑥ クライアントや上司への報告

©SQUARE-ENIX 2011

プロジェクトマネジメントの手順

1. **調査する**
2. **戦略を立てる**（リス
3. **設計する**
4. **計画する**（中長期計画）
   - ① タスクを洗い出す（ユーザーストーリー/フィーチャー/タスク/ＬＯＤ式タスク分解）
   - ② 見積もる（２点見積もり/見積もりポーカー/ＬＯＤ式見積もり）
   - ③ 優先度を決める
   - ④ マイルストーンを定める
5. **スプリント**（４週間単位の制作イテレーション）
   - – 階層化ＰＤＣ
   - ① スプリント計画（成果物目標/スプリント計画会/タスク管理シート）
   - ② 日々の制作（朝会/タスク管理ボード）
   - ③ 週の振り返り（週報/週定例）
   - ④ スプリントの振り返り（スプリント報/スプリント振り返り会/成果物発表会）
   - ⑤ 対策・再計画
     - 戦略・設計の修正（ゲーム仕様、アート、プログラム設計などの修正）
     - 中長期計画の修正（タスクの追加・変更、見積もりの変更、優先度の変更）
     - リソースの修正（人員の追加や担当変更など）
   - ⑥ クライアントや上司への報告

「プロジェクトの立案」や「コンセプトワーク」などは状況によって１～２のどこかで行うことが多いと思います。今回は触れません。

©SQUARE-ENIX 2011

プロジェクトマネジメントの手順

1. **調査する**
2. **戦略を立てる**（リスク一覧/開発戦略マトリクス/価値空間）
3. **設計する**
4. **計画する**（中長期計画）
   - ① タスクを洗い出す（[illegible]分解）
   - ② 見積もる（[illegible]）
   - ③ 優先度を決める
   - ④ マイルス[illegible]
5. **スプリント**（４週間単位の制[illegible]イテレーション）
   - – 階層化ＰＤＣ
   - ① スプリント計画（成果物目標/スプリント計画会/タスク管理シート）
   - ② 日々の制作（朝会/タスク管理ボード）
   - ③ 週の振り返り（週報/週定例）
   - ④ スプリントの振り返り（スプリント報/スプリント振り返り会/成果物発表会）
   - ⑤ 対策・再計画
     - 戦略・設計の修正（ゲーム仕様、アート、プログラム設計などの修正）
     - 中長期計画の修正（タスクの追加・変更、見積もりの変更、優先度の変更）
     - リソースの修正（人員の追加や担当変更など）
   - ⑥ クライアントや上司への報告

今回は主にこの二つにフォーカス

©SQUARE-ENIX 2011

プロジェクトマネジメントの手順

1. **調査する**
2. **戦略を立てる**（リスク一覧/開発戦略マトリクス/価値空間）
3. **設計する**
4. **計画する**（中長期計画）
   - ① タスクを洗い出す（ユーザーストーリー/フィーチャー/タスク/ＬＯＤ式タスク分解）
   - ② 見積もる（２点見積もり/見積もりポーカー/ＬＯＤ式見積もり）
   - ③ 優先度を決める
   - ④ マイルストーンを定める
5. **スプリント**（４週間単位の制作イテレーション）
   - – 階層化ＰＤＣ
   - ① スプリント計画（成果物目標/スプリント計画会/タスク管理シート）
   - ② 日々の制作（朝会/タスク管理ボード）
   - ③ 週の振り返り（週報/週定例）
   - ④ スプリントの振り返り（スプリント報/スプリント振り返り会/成果物発表会）
   - ⑤ 対策・再計画
     - 戦略・設計の修正（ゲーム仕様、アート、プログラム設計などの修正）
     - 中長期計画の修正（タスクの追加・変更、見積もりの変更、優先度の変更）
     - リソースの修正（人員の追加や担当変更など）
   - ⑥ クライアントや上司への報告

©SQUARE-ENIX 2011

調査する①

- すべての起点となります
- 例えば
  - 市場動向を調査する
  - 最新技術動向を調査する
  - スタッフのスキルセットを調査する
  - 会社の財務状況を調査する
  - リスク要因を調査する
  - 他プロジェクトの状況を調査する
  - 過去のプロジェクトの予算と実績を調査する

※各国の経済状況予測までしないとならない時代に突入したかもしれません

©SQUARE-ENIX 2011

調査する②

- 戦略や設計や計画や制作をスムーズに進め、なるべく手戻りを少なくするためにとても大切なアクションです
- 類似商品を発掘し尽くし、それらを徹底的に要素分解することも含まれます
- リスクを減らすために出来る限りの調査をしましょう
- 今回は調査の話は軽めに終えます

©SQUARE-ENIX 2011

プロジェクトマネジメントの手順

1. **調査する**
2. **戦略を立てる**（リスク一覧表/開発戦略マトリクス/価値空間）
3. **設計する**
4. **計画する**（中長期計画）
   - ① タスクを洗い出す（ユーザーストーリー/フィーチャー/タスク/ＬＯＤ式タスク分解）
   - ② 見積もる（２点見積もり/見積もりポーカー/ＬＯＤ式見積もり）
   - ③ 優先度を決める
   - ④ マイルストーンを定める
5. **スプリント**（４週間単位の制作イテレーション）
   - – 階層化ＰＤＣ
   - ① スプリント計画（成果物目標/スプリント計画会/タスク管理シート）
   - ② 日々の制作（朝会/タスク管理ボード）
   - ③ 週の振り返り（週報/週定例）
   - ④ スプリントの振り返り（スプリント報/スプリント振り返り会/成果物発表会）
   - ⑤ 対策・再計画
     - 戦略・設計の修正（ゲーム仕様、アート、プログラム設計などの修正）
     - 中長期計画の修正（タスクの追加・変更、見積もりの変更、優先度の変更）
     - リソースの修正（人員の追加や担当変更など）
   - ⑥ クライアントや上司への報告

©SQUARE-ENIX 2011

戦略を立てる

- 細かな設計や計画に入る前に、担当プロジェクトの開発戦略を入念に考えることで後々のリスクを最小化します
  - 「リスク一覧表」の作成（調査フェイズで終わっていることが理想）
  - 「開発戦略マトリクス」を参考に意志決定を予め行う
  - 「商品の価値空間」を意識して、プロジェクトに本当に必要な要素は何かをしっかり考える
- ゲームデザイン、アートワーク、プログラム、サウンド他、すべての項目が対象です

©SQUARE-ENIX 2011

リスク一覧表

項目	難易度	重要度	警戒度 (難×重)	対策案	責任者
○○機能	2	4	8	・・・	Aさん
△△システム	5	5	25	・・・	Bさん
村	3	2	6	・・・	Aさん
・・・	・・・	・・・	・・・	・・・	・・・

- さまざまな事前調査を受けて、案件別に難易度と重要度のスコアをつけてみます
- 難易度と重要度のスコアを掛けたものを警戒度とします
- 警戒度が高いものを中心に予め入念に対策を練っておきます
  - 代替案を用意してでも絶対になにかを実現するのか、最悪切り捨てても問題ないのか、いつ、どうやって「安全化」するのかを考えます
- 対策責任者も明確にします
- ゲームデザイン、アートワーク、プログラム、すべてが対象です

©SQUARE-ENIX 2011

開発戦略マトリクス

重要度 ＼ 難易度	易		難
	必ず作れる要素	おそらく作れる要素	作ってみないと不明
高	必要な要素	・優先度中★★★ ・迷わず作る ・他に影響を与えないのならば、むしろ後回しでもOK	・優先度高★★★★ ・できるだけ早期に検証・実験する	・優先度最高★★★★★ ・最優先で検証・実験し、バックアッププランを用意
	できれば欲しい要素	・優先度低★★	・優先度中★★★ ・優先的に検証・実験するかどうかを早期に決める ・優先度を下げる場合は切り離し可能な状態を維持してゆとりの人員と時間で挑戦	・優先度低★★ ・なるべく作らない ・作る場合は切り離し可能な状態を維持してゆとりの人員と時間で挑戦
低	無くても困らない要素	・優先度最低★ ・人員や時間にとても余裕があったら後で作る	・優先度最低★ ・なるべく作らない ・とても余裕があったら挑戦してみても良い	・優先度× ・作ってはならない

※あくまで概念の紹介として記載していますので各方針や優先度の記述は例として捉えてください
※いろいろな事情や戦略によっては優先度は変化します

©SQUARE-ENIX 2011

開発戦略マトリクス

- 必ず作れる物、たぶん作れる物、やってみないと出来るかどうか分からない物などに分類します
- 本当に必要な物、できれば欲しいもの、無くても困らない物などに分類します
- マトリクスを参考に、やること、やらないこと、バックアッププラン、優先順位などを決めます
- ゲームデザイン、アートワーク、プログラム、すべてが対象です

©SQUARE-ENIX 2011

商品の価値空間

- 通常は価値＝質×量と簡単なモデル化をすることがありますが、本手法では

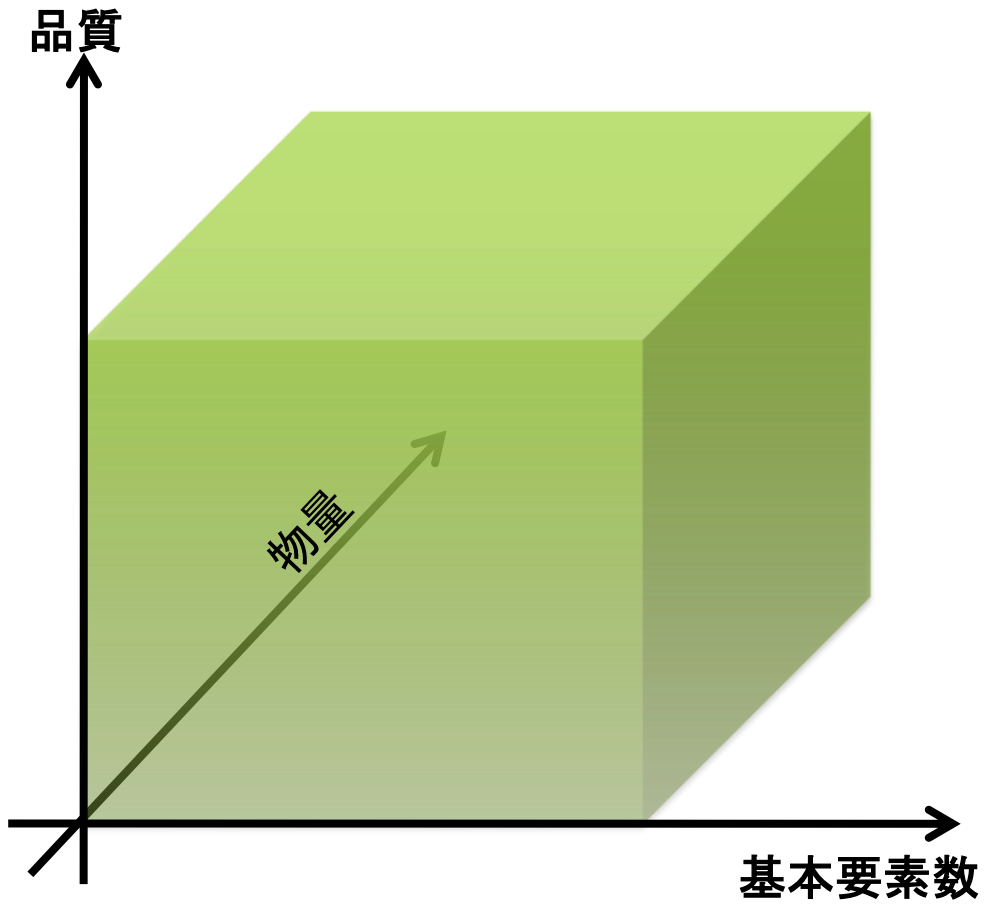
  価値＝基本要素数×品質×物量

  と設定します

©SQUARE-ENIX 2011

商品の価値空間

- **品質（高さ）**
  - 面白さ
  - 操作性
  - 映像の美しさ、カッコよさ
  - 魅力的な世界観と登場キャラクター
  - 物語の展開
  - エピックな体験
  - 快適さ
  - 分かりやすさ
- **基本要素数（幅）**
  - アクションパート
  - タウンパート
  - ミニゲームパート
  - 収集、効果
  - 育成
  - イベント
  - オンライン
  - メニュー
  - 技術チャレンジ
  - ○○システム
- **物量（奥行き）**
  - ステージ数
  - エネミーの種類
  - 武器の種類
  - イベントの数



※各軸は完全な直交性はありませんが、概念の理解の為にあえて分離しています
※体積が大きいからといって必ずしも価値が高いとは限らないですが、目安として見てください

©SQUARE-ENIX 2011

商品の価値空間

- **基本要素数（幅）**
  - 純粋な物量を取り除いた、基本要素の種類数を指します
  - 敵、ボス、空を飛ぶ、コンボシステム、時間の変化、天候の変化、オープンワールド、カットシーン、などの単位です
  - 面白さ、或いは技術面でなにかしら乗り越えないとならないものが多く含まれ、通常このファクターにリスクが集中します
  - 質でもあり量でもある部分です
  - 不確実性を減らす場合に最も効果を発揮するポイントであり、ここの種類を減らすことでスケジュールの見通しは立ちやすくなります
  - 「ほんとうにその基本要素は必要なのか？」という問いをプロジェクト初期に真剣に行いましょう
- **品質（高さ）**
  - 基本要素に対しての質です
  - 基本要素の構築を乗り越えていれば、単純に作り込みによって高まるファクターです
  - コントロールしやすい要素です
- **物量（奥行き）**
  - 基本要素に対しての量産の数です
  - 基本要素の構築を乗り越えていれば、単純に時間や人数によって増やせるファクターです
  - コントロールしやすい要素です

©SQUARE-ENIX 2011

戦略を立てる

- これらの考え方や他の考え方などを組み合わせてリスク分析、価値分析、状況分析などをして開発戦略を立てます
- 今回は戦略の話は軽めに終えます

©SQUARE-ENIX 2011

プロジェクトマネジメントの手順

1. **調査する**
2. **戦略を立てる**（リスク一覧/開発戦略マトリクス/価値空間）
3. **設計する**
4. **計画する**（中長期計画）
   - ① タスクを洗い出す（ユーザーストーリー/フィーチャー/タスク/ＬＯＤ式タスク分解）
   - ② 見積もる（２点見積もり/見積もりポーカー/ＬＯＤ式見積もり）
   - ③ 優先度を決める
   - ④ マイルストーンを定める
5. **スプリント**（４週間単位の制作イテレーション）
   - – 階層化ＰＤＣ
   - ① スプリント計画（成果物目標/スプリント計画会/タスク管理シート）
   - ② 日々の制作（朝会/タスク管理ボード）
   - ③ 週の振り返り（週報/週定例）
   - ④ スプリントの振り返り（スプリント報/スプリント振り返り会/成果物発表会）
   - ⑤ 対策・再計画
     - 戦略・設計の修正（ゲーム仕様、アート、プログラム設計などの修正）
     - 中長期計画の修正（タスクの追加・変更、見積もりの変更、優先度の変更）
     - リソースの修正（人員の追加や担当変更など）
   - ⑥ クライアントや上司への報告

©SQUARE-ENIX 2011

設計する①

- 計画や制作をスムーズに進め、なるべく手戻りを少なくするためにとても大切なアクションです
- 例えば
  - ゲーム仕様を設計する
  - キャラクターや背景のアートワークを用意する
  - 世界観設定を用意する
  - シナリオのプロットを考える
  - 技術戦略を考える
  - プログラム設計をする

©SQUARE-ENIX 2011

設計する②

- 原則、頭の中で想像の出来得る限りは設計をとことん行い、思考検証し、設計を洗練させます
- **作らずとも頭の中で分かることはまず先に設計し尽くすべきです**
- 実際に作って物にしてみないと分からないことがあって初めて**「設計のための仮実装（＝実験）」**を行うべきです

©SQUARE-ENIX 2011

設計する③

- ひとり～数人で数カ月で作るゲームならほぼ無計画に作ってもさほど問題無いでしょう。
- しかし、数十人～数百人、年単位で作るゲームで基本設計が固まっていないままプロジェクトを本格稼働させるのは大変危険です
- **特にゲーム仕様、技術仕様は早期に固めましょう**
- 今回は設計の話は軽めに終えます

©SQUARE-ENIX 2011

プロジェクトマネジメントの手順

1. **調査する**
2. **戦略を立てる**（リスク一覧/開発戦略マトリクス/価値空間）
3. **設計する**
4. **計画する**（中長期計画）
   - ① タスクを洗い出す（ユーザーストーリー/フィーチャー/タスク/ＬＯＤ式タスク分解）
   - ② 見積もる（２点見積もり/見積もりポーカー/ＬＯＤ式見積もり）
   - ③ 優先度を決める
   - ④ マイルストーンを定める
5. **スプリント**（４週間単位の制作イテレーション）
   - – 階層化ＰＤＣ
   - ① スプリント計画（成果物目標/スプリント計画会/タスク管理シート）
   - ② 日々の制作（朝会/タスク管理ボード）
   - ③ 週の振り返り（週報/週定例）
   - ④ スプリントの振り返り（スプリント報/スプリント振り返り会/成果物発表会）
   - ⑤ 対策・再計画
     - 戦略・設計の修正（ゲーム仕様、アート、プログラム設計などの修正）
     - 中長期計画の修正（タスクの追加・変更、見積もりの変更、優先度の変更）
     - リソースの修正（人員の追加や担当変更など）
   - ⑥ クライアントや上司への報告

©SQUARE-ENIX 2011

計画の手順

① タスクを洗い出す
② 見積もる
③ 優先度を決める
④ マイルストーンを定める

©SQUARE-ENIX 2011

計画の手順

① タスクを洗い出す
② 見積もる
③ 優先度を決める
④ マイルストーンを定める

©SQUARE-ENIX 2011

計画の手順

① タスクを洗い出す
   A) ユーザーストーリーの列挙
   B) フィーチャーに分解
   C) タスクに分解
② 見積もる
③ 優先度を決める
④ マイルストーンを定める

©SQUARE-ENIX 2011

ユーザーストーリーとは①

- そのソフトウェア（ゲーム含む）で実現したいことを「文章で」表現したものです
- ユーザーストーリーと図解を組み合わせて「仕様書」としても良いです
- 例
  - １００km四方のフィールドを自由に散策できる
  - プレイヤーも敵も建物を破壊することができる
  - プレイヤーはライフがゼロになったら死んでリスタートポイントで復帰できる
  - 敵キャラクターは５０人同時に画面に登場する
  - プレイヤーはほふく前進をして、狭い隙間をくぐりぬけることができる
  - 煙がキャラクターをよける動きをする
  - スイッチを踏んだらドアが開く

©SQUARE-ENIX 2011

ユーザーストーリーとは②

- とにかく実現したいこと、すべきことを漏らさず洗いつくすことに注力します
- マインドマップソフトを活用するのがオススメです
- 複数人で宿題でユーザーストーリーを洗い出したマインドマップを持ち寄って、大きなマインドマップを作ると漏れを少なくしやすいです

©SQUARE-ENIX 2011

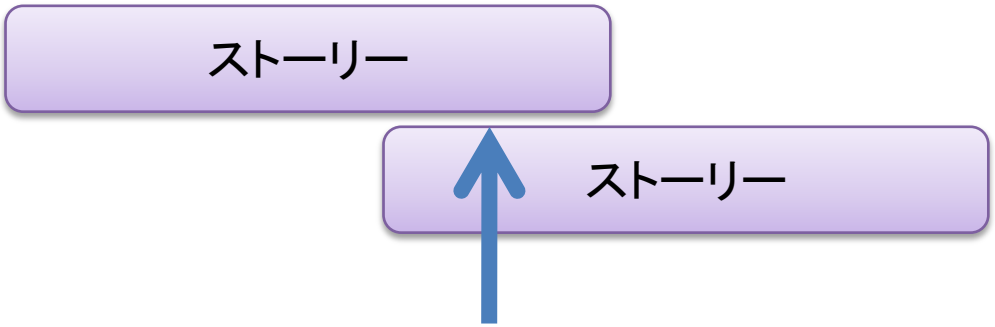
ユーザーストーリーとは③

- 次のステップのフィーチャーを列挙する手掛かりとなるのがユーザーストーリーです
- このユーザーストーリーの範囲こそがスコープ（作る範囲）となります
- 制作が進んできた際の品質のガイドラインともなります
- ユーザーストーリーが大量に漏れていた場合、後で大きな手戻りが発生する可能性があるので極力最終形の８割ほどは網羅することを目標に洗い出します

©SQUARE-ENIX 2011

ユーザーストーリーとは④

- 書物によっては「重なり合わないこと」とありますが、重なり合うことは寧ろアリと考えます
- 漏れが無いことを重視します



重なり歓迎！

©SQUARE-ENIX 2011

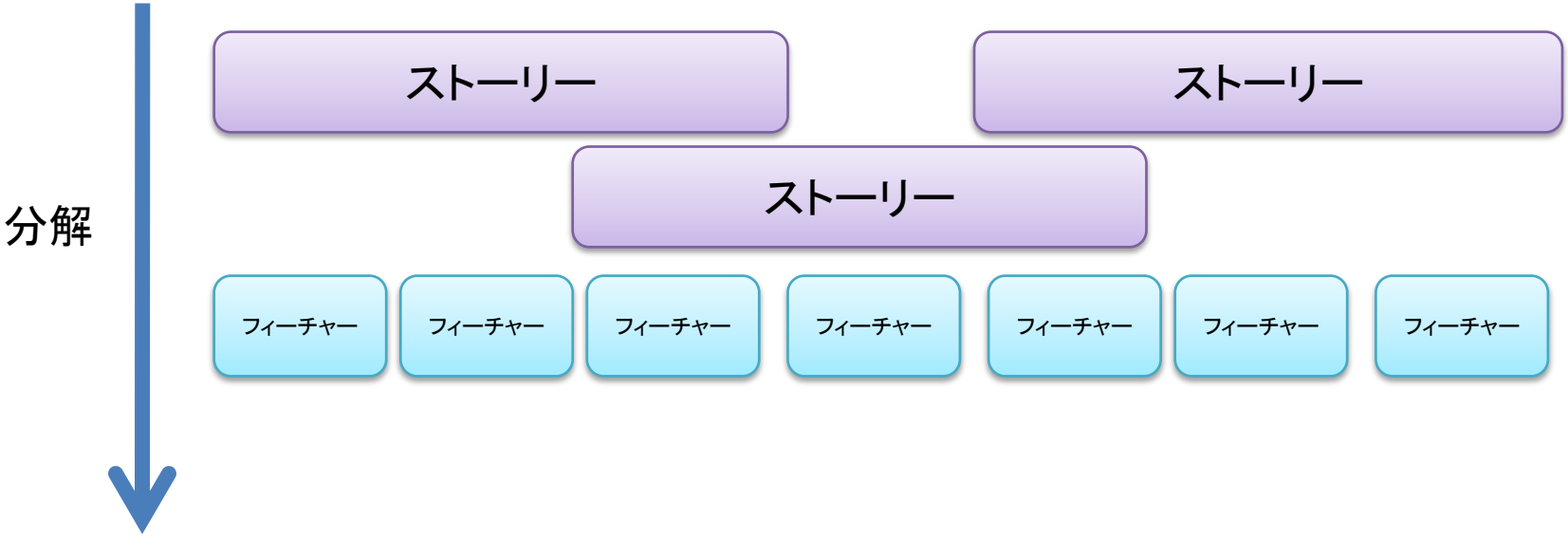
フィーチャーとは①

- 制作する機能や対象を意味します
- ユーザーストーリーを分解することで列挙します
- ○○機能、○○システムなどの「名詞で終わる」事が多いです
- 例
  - ライン描画機能
  - プレイヤーのジャンプ
  - プレイヤーのほふく前進
  - 炎攻撃
  - メモリ管理システム

©SQUARE-ENIX 2011

フィーチャーとは②

- 「大きなタスク」と言っても良いです
- 別々のストーリーを分解してみると、その重なりから同じフィーチャーが出てくることは普通です



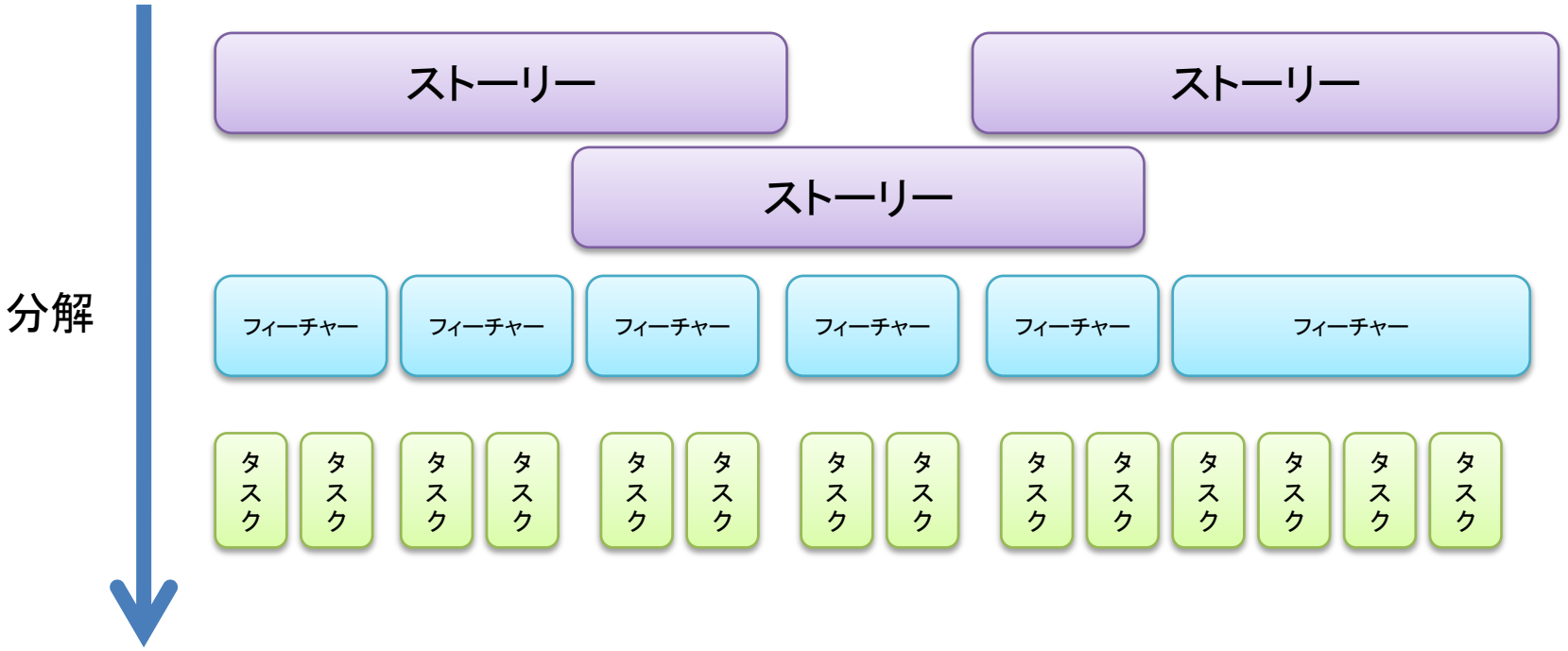
©SQUARE-ENIX 2011

タスクとは

- 各メンバーが作業を行う管理単位です
- フィーチャーを分解した物です
- １日あたり１～２タスク程度のサイズまで分解します
- 例
  - スコア表示のフェードイン、フェードアウト処理
  - ほふく前進モーション
  - ドアの開閉

©SQUARE-ENIX 2011

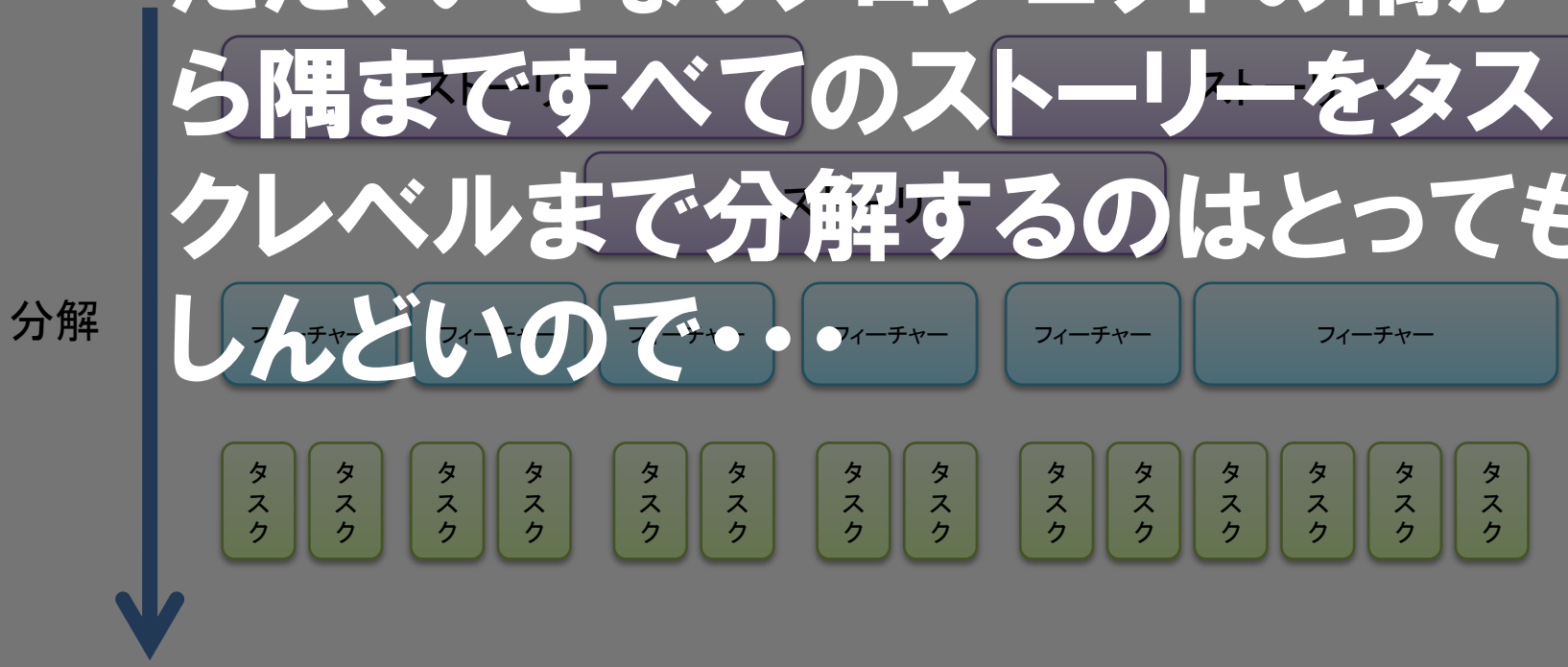
ストーリーとフィーチャーとタスク



©SQUARE-ENIX 2011

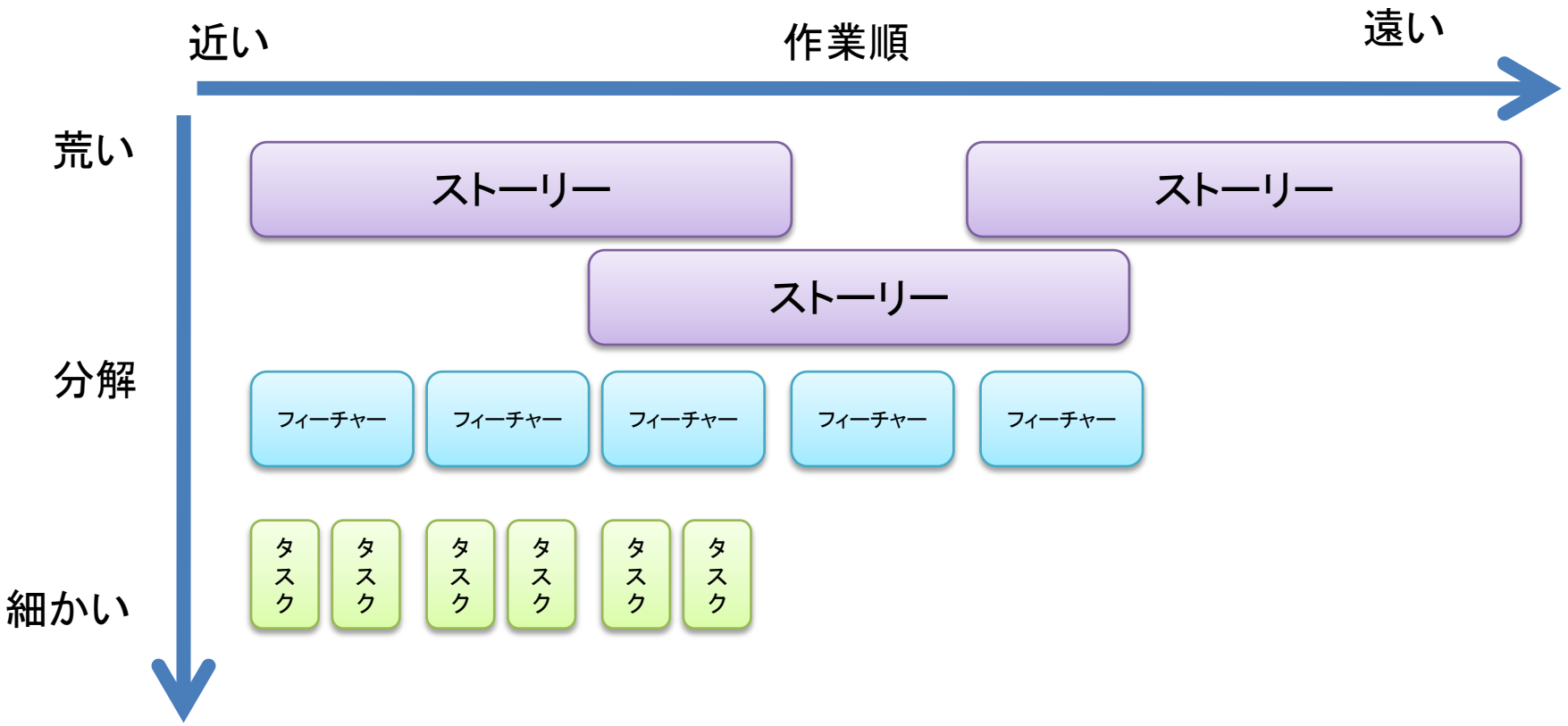
ストーリーとフィーチャーとタスク

ただ、いきなりプロジェクトの隅から隅まですべてのストーリーをタスクレベルまで分解するのはとってもしんどいので・・・



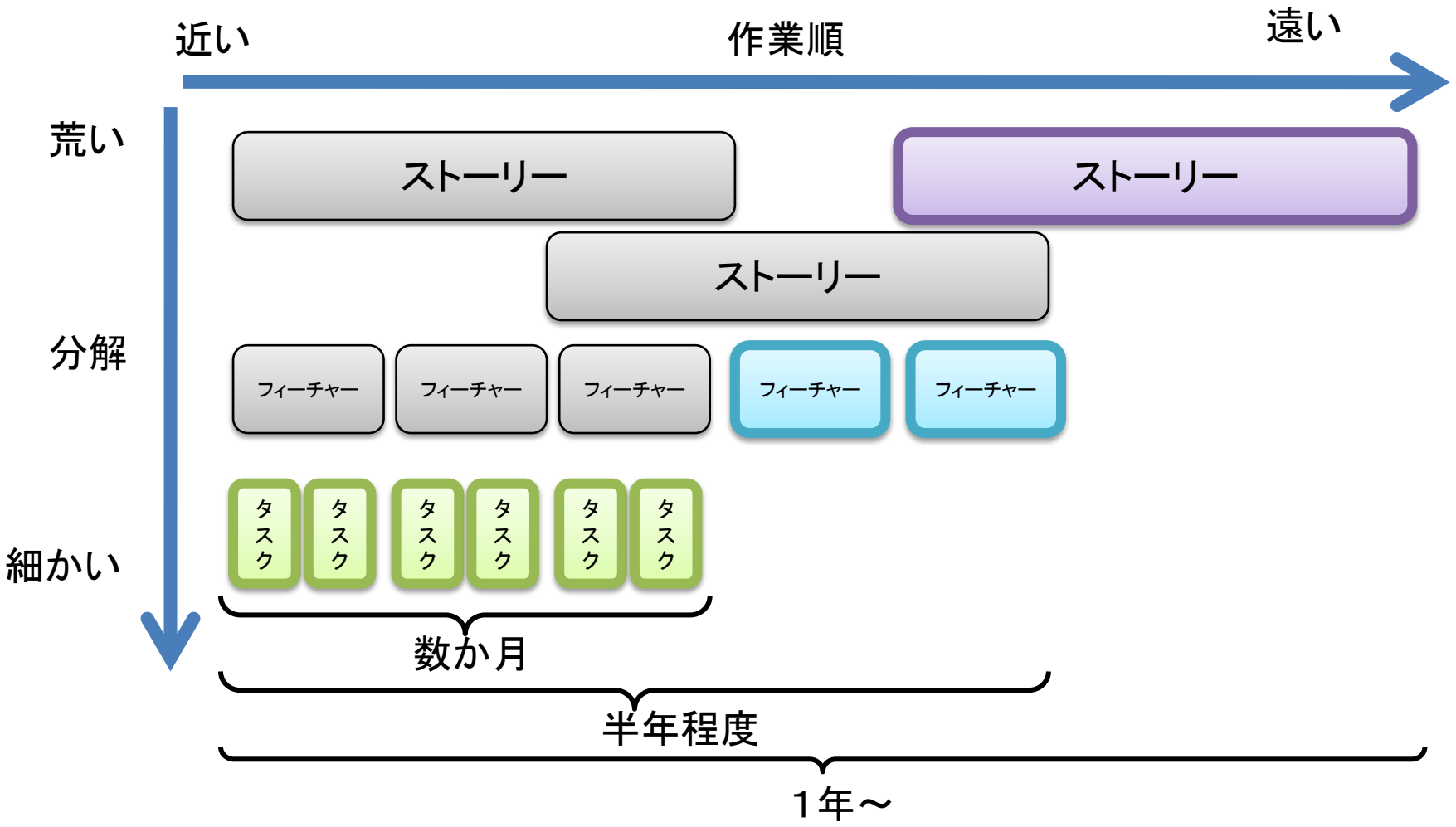
©SQUARE-ENIX 2011

LOD的分解で時間効率を上げる



©SQUARE-ENIX 2011

LOD的分解で時間効率を上げる



©SQUARE-ENIX 2011

計画の手順

① タスクを洗い出す
   A) ユーザーストーリーの列挙
   B) フィーチャーに分解
   C) タスクに分解
   – LOD式タスク分解
② 見積もる
③ 優先度を決める
④ マイルストーンを定める

©SQUARE-ENIX 2011

計画の手順

① タスクを洗い出す
   A) ユーザーストーリーの列挙
   B) フィーチャーに分解
   C) タスクに分解
   – LOD式タスク分解
② 見積もる
   – ２点見積もり
   – 見積もりポーカー
   – LOD式見積もり
③ 優先度を決める
④ マイルストーンを定める

©SQUARE-ENIX 2011

見積もり

- 列挙したタスク群に対してそれぞれ何日かかるのかを見積もりします
- 以下の見積もりのテクニックを用います
  - ２点見積もり
  - 見積もりポーカー
  - LOD式見積もり

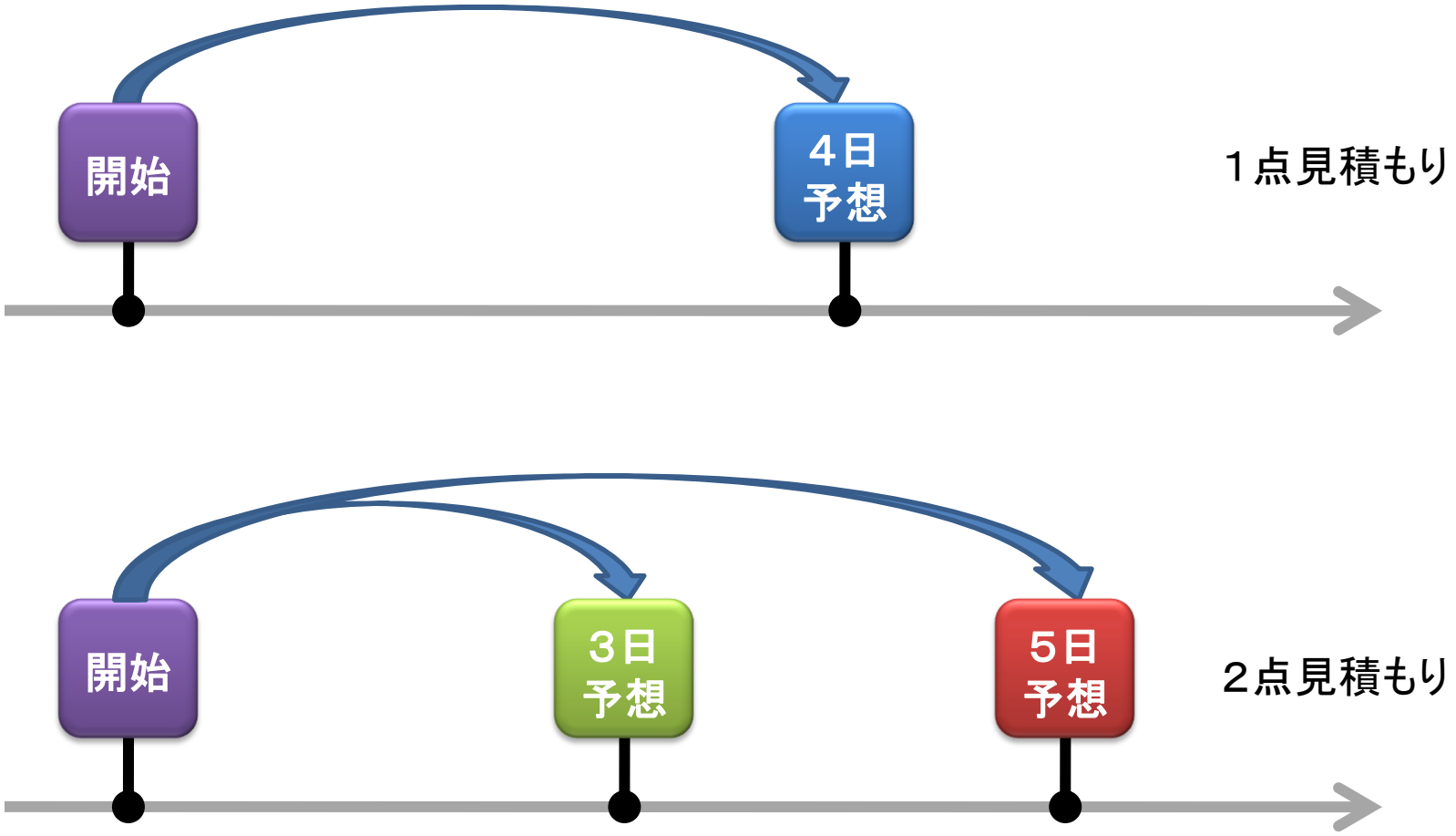
©SQUARE-ENIX 2011

２点見積もり

- 通常は作業日数をひとつだけ予想する「１点見積もり」を行うケースが多いですが、本手法では二つの日数を予想する「２点見積もり」という方法を行います
- ○時間～○時間、あるいは○日～○日と二つの数字で表現します
- ２点見積もりは心理力学の観点で非常に効果的でモチベーションの維持に向いていますし、分析時にも大変役に立ちます

©SQUARE-ENIX 2011

１点見積りと２点見積もり



©SQUARE-ENIX 2011

１点見積りの問題点

- １点見積もりはメジャーな方法ですが、問題や欠点が多いです
- 具体的には以下の悪い現象が発生します
  - 定義のあいまいさ
  - パーキンソンの法則
  - 学生症候群
  - のど元過ぎれば熱さ忘れるの法則

©SQUARE-ENIX 2011

定義のあいまいさ

- １点見積もりで「４日」という予想があったとして、定義があいまい故にその「４日」の意味が人によって異なってしまいます
  - ある人は4日で出来上がるかどうか、実際に必要な日数と同じかそれよりも小さめのギリギリの日数を見積もります
  - 別の人は本当は3日で終わると思っているのですが安全をみてプラス1日して4日と見積もります
  - あるいは1日で終わると思っているのに4日という人もいるかもしれません
- 定義があいまいなことで
  - 皆から集めた見積もりの一覧の精度は大変荒いものになってしまいます
  - 元々の意味がバラバラなので、進捗管理を行う際に消化日数の意味も不明になり、数値からリスクを読み取ることが難しくなります

©SQUARE-ENIX 2011

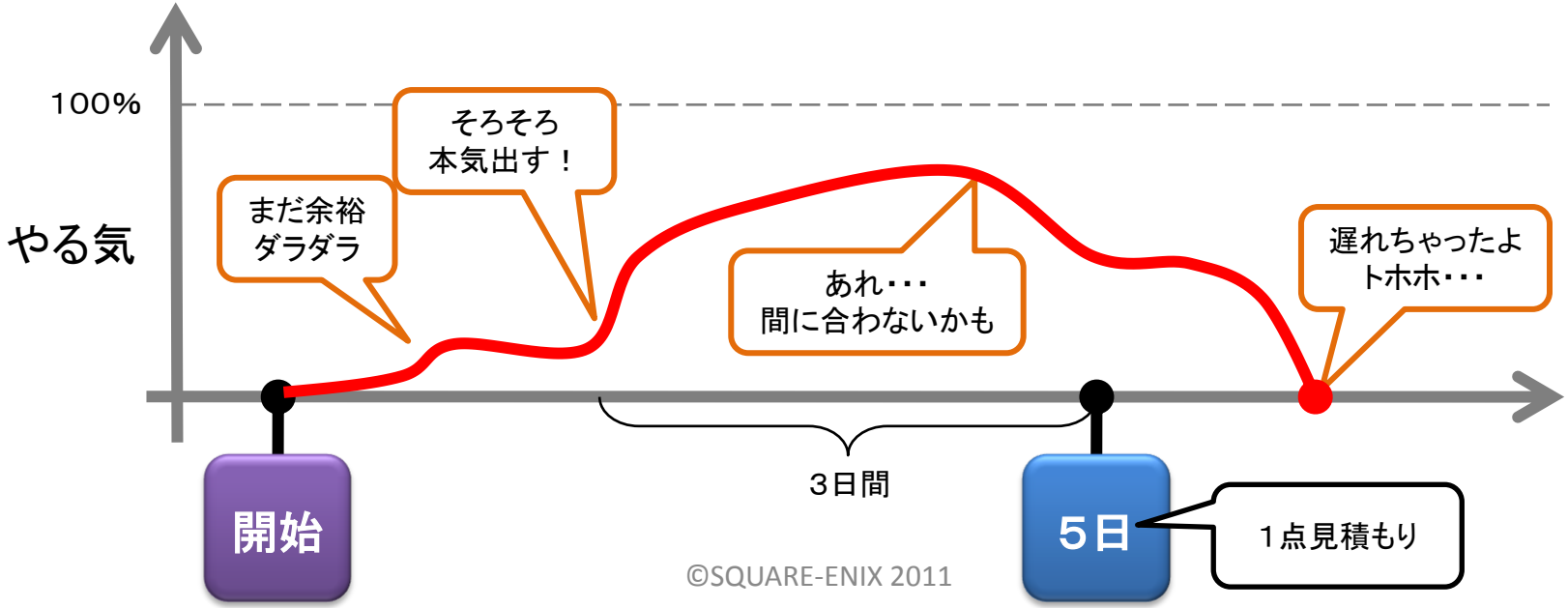
パーキンソンの法則

- 「仕事は、その遂行のために利用できる時間をすべて埋めるように拡大する」という現象
- ４日という１点見積もりをしていて、２日で片付いてしまったとしても、完了報告をせずにのんびり過ごしたり、ギリギリまで必要以上の作業をしてしまうタイプの人がいます
- なのでバッファを積んでも積んでも消費しつくしてしまいます

©SQUARE-ENIX 2011

学生症候群／夏休みの宿題病

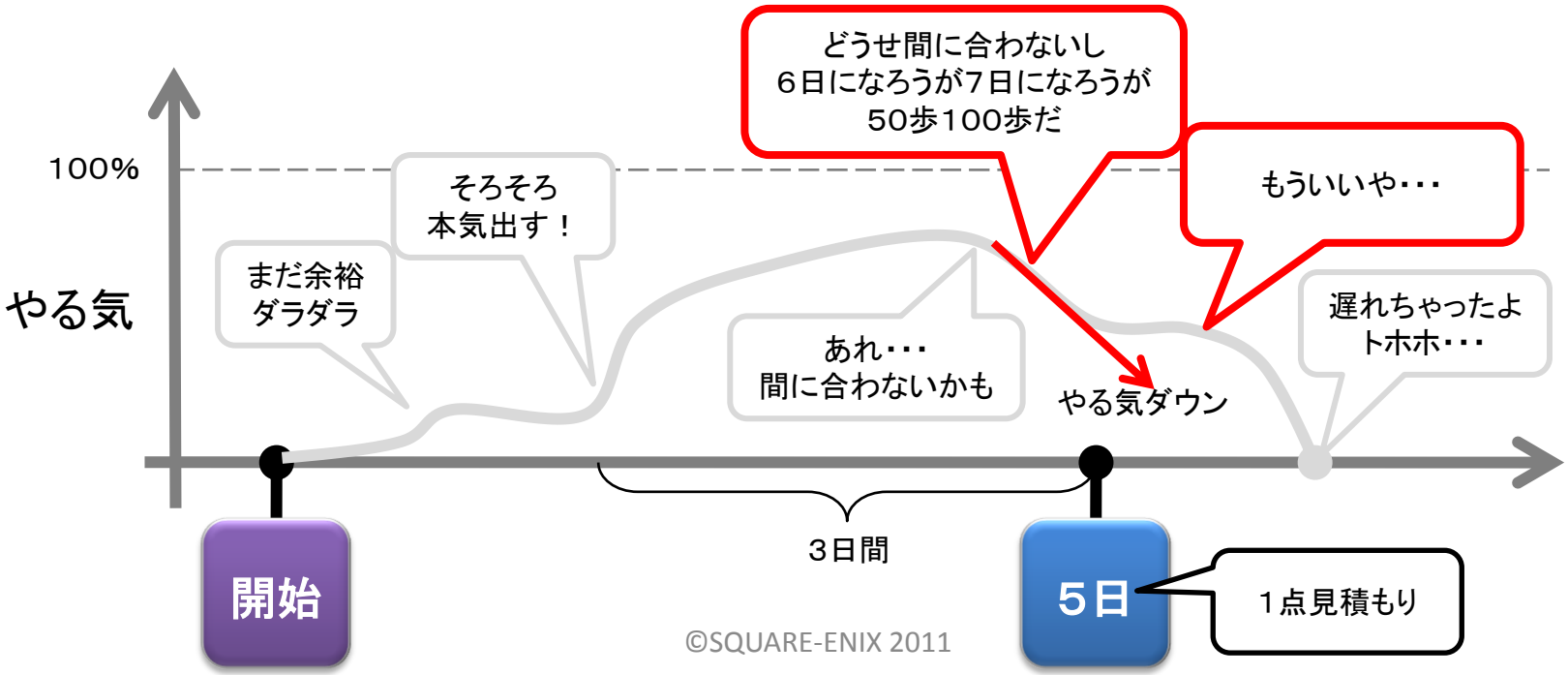
- 時間に余裕がある場合に作業の開始をギリギリまで遅らせてしまう性質の事を言います
- 典型パターン
  - 「５日」と１点見積もりしたとして、実はそのうち２日をバッファとして想定していたとします
  - この人は3日もあれば作業が終わると思っているので（人によっては）最初の２日はダラダラ過ごします
  - 残り３日となって「そろそろ本気出す！」と気合を入れ始めます
  - ところが３日という事前予想が当たる事の方が珍しいわけで、結構な確率で3日よりも日数が必要になります
  - 結果当初の見込みの５日よりも時間が必要になってしまいます



©SQUARE-ENIX 2011

のど元過ぎれば熱さ忘れるの法則

- 一旦スケジュールに間に合わないことが分かると集中力が落ちてしまう性質の事です
- 典型パターン
  - さきほどの学生症候群と同じ図で説明します
  - 「そろそろ本気出す！」と気合モードに入ってから、その気合いは「このままでは見積もり日程内に収まらない・・・」と途中で判明するまでは維持できるケースが多いのですが、間に合わないと一旦判明してしまうと「どうせ間に合わないのは一緒」となってしまい、１日でも半日でも１時間でも急ごうとする気持ちが萎えてペースが落ちてしまう現象が（人によっては）発生します



©SQUARE-ENIX 2011

１点見積りの問題点

- １点見積もりには以下のような問題点がありました
  - 定義のあいまいさ
  - パーキンソンの法則
  - 学生症候群
  - のど元過ぎれば熱さ忘れるの法則

©SQUARE-ENIX 2011

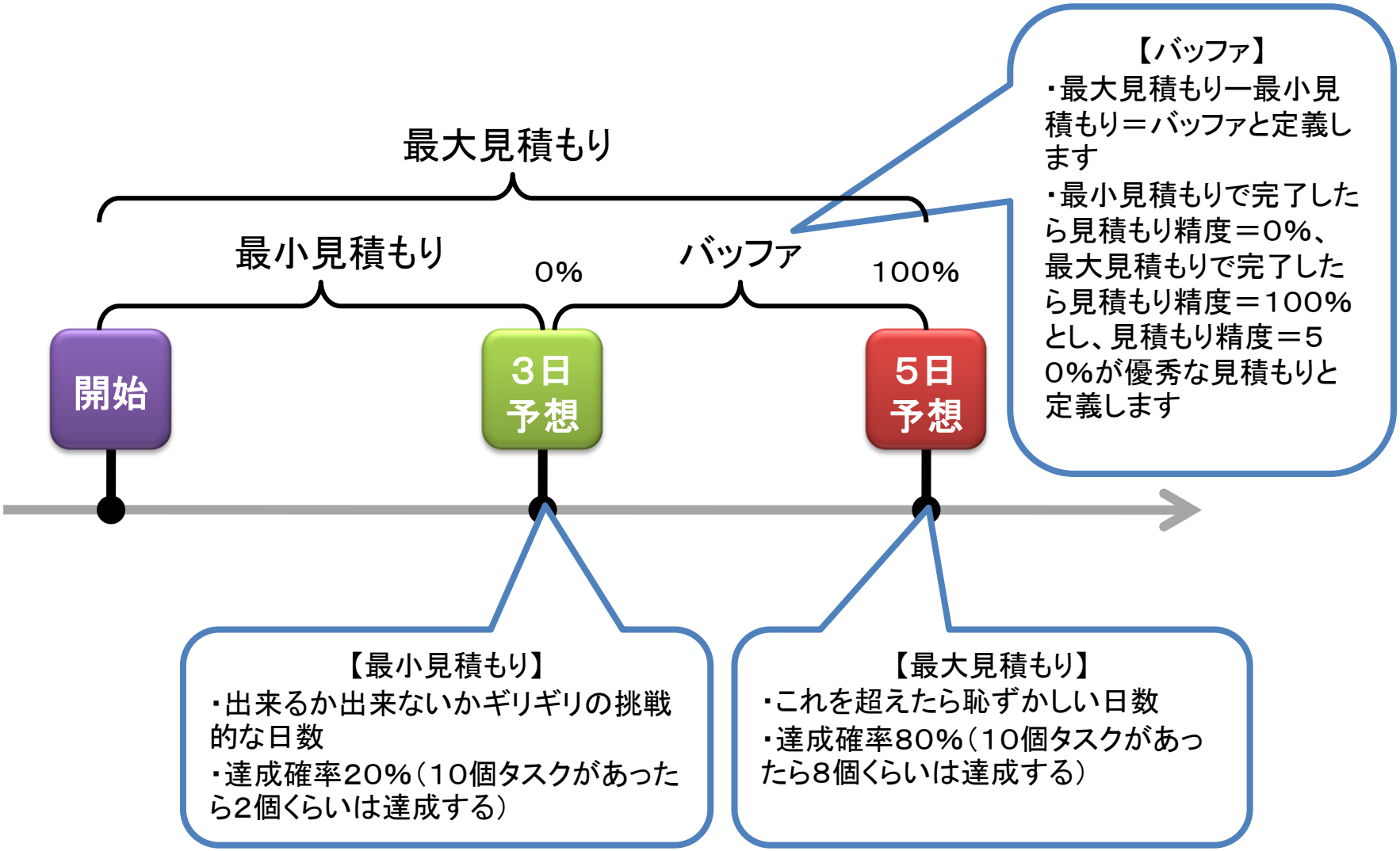
１点見積りの問題点

- １点見積もりには以下のような問題点がありました
  - 定義のあいまいさ
  - パーキンソンの法則
  - 学生症候群
  - のど元過ぎれば熱さ忘れるの法則

なんと2点見積りではこれらを心理力学として解消できます！

©SQUARE-ENIX 2011

2点見積もり



©SQUARE-ENIX 2011

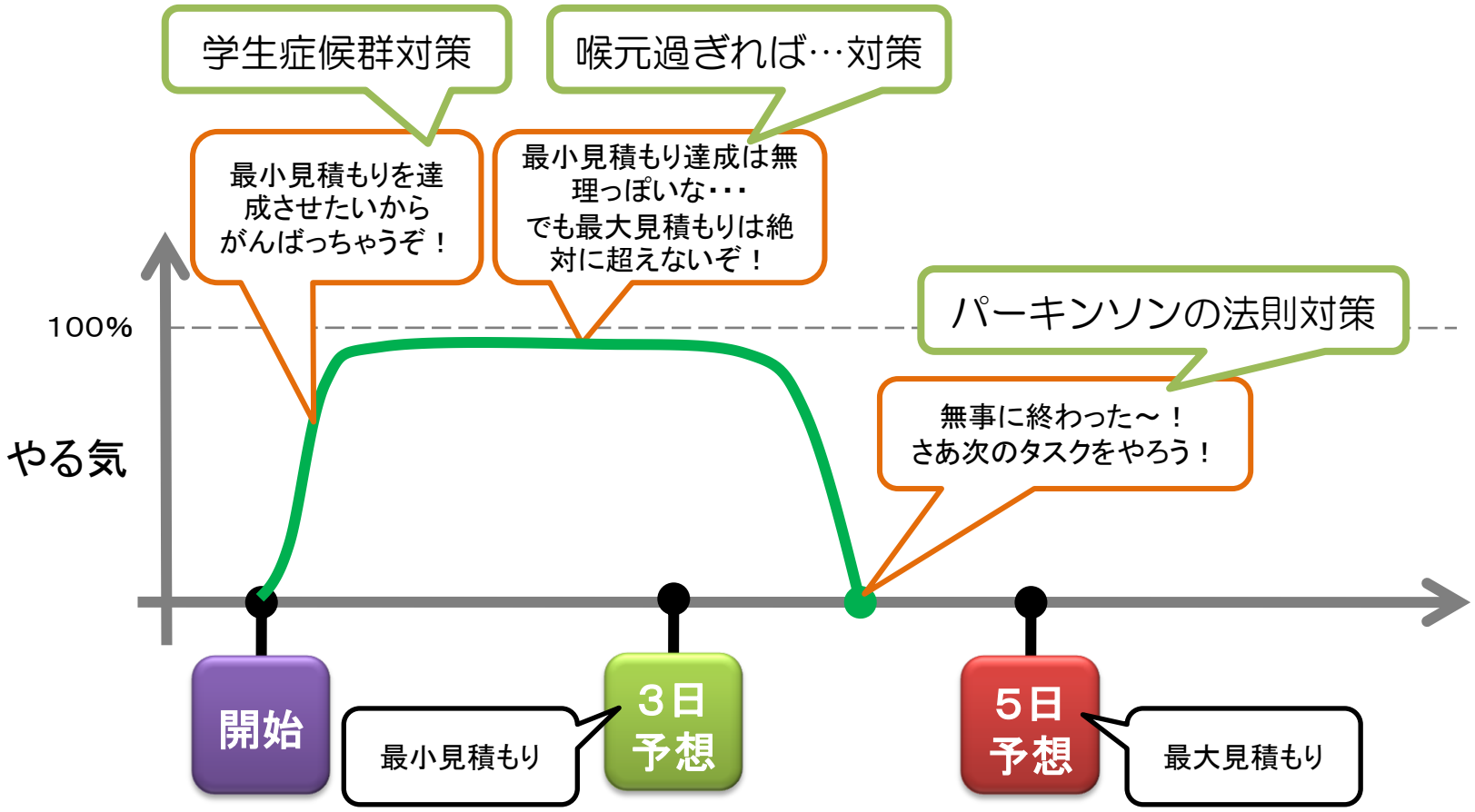
２点見積りのメリット

- ２点見積もりは１点見積もりの欠点を補います
  - 定義のあいまいさ⇒OK
    - 最小見積もり、最大見積もりという定義が入るので、個人による解釈の差が生まれにくいです
  - パーキンソンの法則⇒OK
    - 「見積もり精度」がタスク毎に表現されるので、タスクが片付いたらそのタスクをいつまでも抱えることなく、さっさと次のタスクを開始することに喜びを感じます
  - 学生症候群⇒OK
    - 一般的に、難しいのが分かっていても人は最小見積もりを目指す傾向があるので、いきなりアクセル全開で作業を開始してくれます
  - のど元過ぎれば熱さ忘れるの法則⇒OK
    - 最小見積もりを超えるのは普通の事ですが、例え最小見積もりを超えてもさらに最大見積もりという「超えたら恥ずかしい、絶対に死守したい防衛ライン」が控えているので、引き続き心が萎える事無くアクセル全開状態を維持できます

©SQUARE-ENIX 2011

２点見積りのメリット

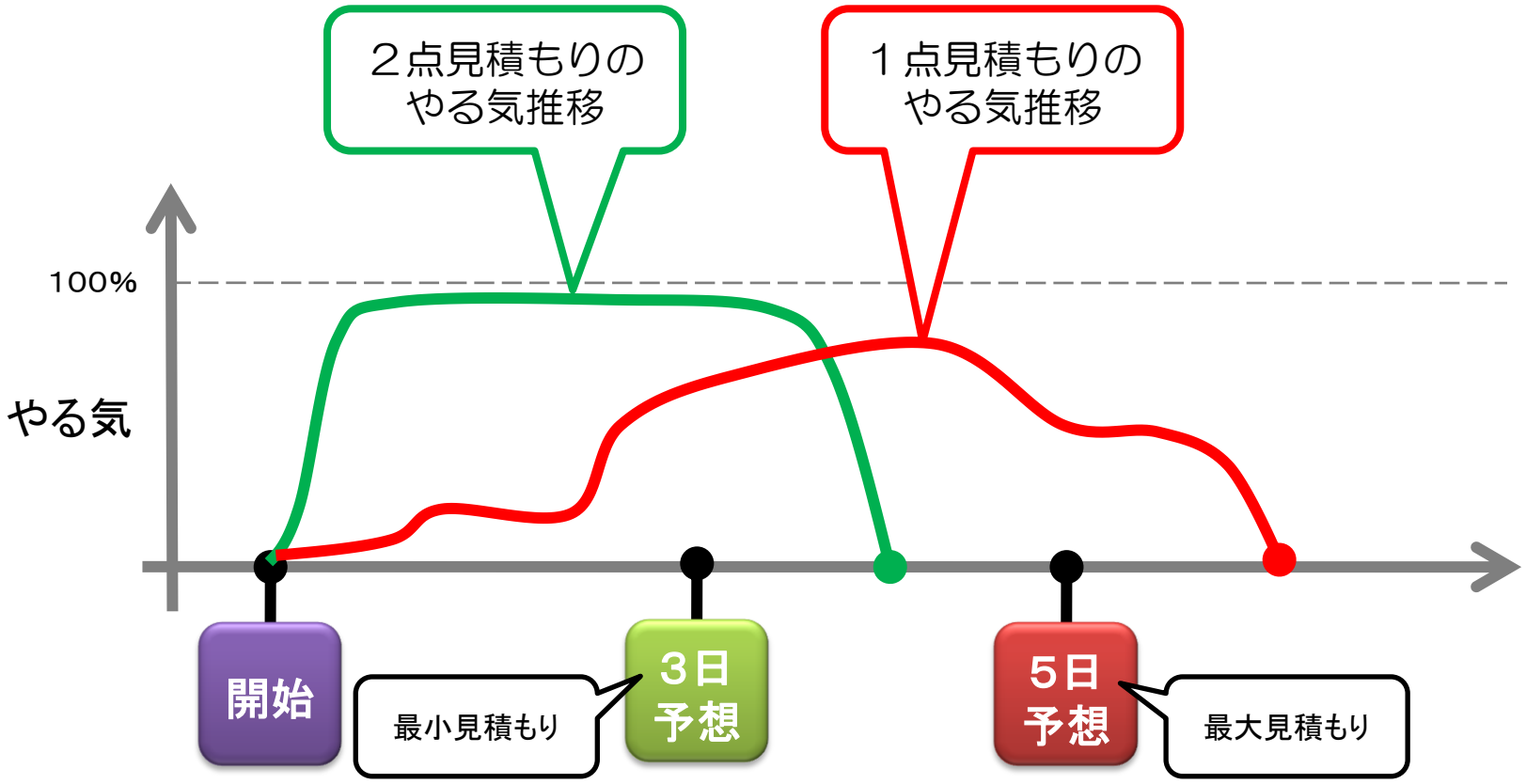
- ２点見積もりは、１点見積もりの数々の欠点を解消します



©SQUARE-ENIX 2011

２点見積りのメリット

- 心理作用により、同じ能力、同じ性格の人が作業をしても、１点見積もりを採用するか、２点見積もりを採用するかで全く結果が異なってしまうのです



©SQUARE-ENIX 2011

３点見積りではだめなのか

- 「最小見積り」「最大見積り」「目標見積り」という３つの日数を予想し、加重平均により完了確率の高い予測日を算出する方法があります
- ３点見積もりを行うと、心の中で「目標見積り」の日数、或いは加重平均日の存在が大きくなり、そこに意識がフォーカスされ、心理的には実質１点見積もりを行っているのと同じ状態に陥り易くなります
- 手間が増える割に得よりも損の方が大きいと考えています

©SQUARE-ENIX 2011

見積もりポーカー①

- 見積もりを行う際には見積もりポーカーというものを使います
- 見積もるタスクリストの作業担当者、その同僚、上司などの３人ほどで集まって、タスクリストの各タスクを見積もることで、精度の向上と、タスクの相互理解を促します
- やってみるとわかりますが面白いです（聞くのとやるのとでは違います、実際にやってみてください）
- 世間的には１枚のカードを場に出すのが普通ですが、私達の手法では最小、最大の２枚のカードを場に出します

©SQUARE-ENIX 2011

見積りポーカー②

- 無地のトランプを使って見積り専用のカードを作成する（雑貨屋の手品師コーナーなどで買えます）
- 見積りは３人程度で行う（担当者と同僚と上司など）
- 見積りを行うタスクリストを用意する
- タスクリストから見積りをするタスクを選ぶ
- タスクの内容を見積りメンバー間で誤解の無いように確認する
- それぞれでそのタスクに何日必要か考える
- 「せーの」でそれぞれ最小見積り日数と最大見積り日数のカードを２枚出す
- 食い違った場合はなぜその見積りなのかを互いに説明し合う
- カードをそれぞれ回収
- 見積りがすり寄るまで繰り返す（だいたい２～３周ですり寄る）
- タスクリストのタスクすべてが見積られるまで繰り返す

©SQUARE-ENIX 2011

見積もりもLOD式で

- 見積もりも、タスク分解と同様にいきなりすべての詳細なタスクに対して見積もり作業を行うと日が暮れてしまうので、うまく省略しながら行います
- 近い作業はタスクのレベルで見積もり、遠いタスクはフィーチャーやストーリーのレベルで見積もりをします
- 詳しくは別の機会にでも

©SQUARE-ENIX 2011

計画の手順

① タスクを洗い出す
   A) ユーザーストーリーの列挙
   B) フィーチャーに分解
   C) タスクに分解
   – LOD式タスク分解
② 見積もる
   – ２点見積もり
   – 見積もりポーカー
   – LOD式見積もり
③ 優先度を決める
④ マイルストーンを定める

©SQUARE-ENIX 2011

優先度を決める

- 詳細なタスクが列挙され、それが見積もられているならば、それらのタスクの処理の優先度を決めればそれがスケジュール表となります
- 実際にはユーザーストーリーの段階で優先度が付き、フィーチャーの段階で優先度が付き、タスクの段階で優先度が付き・・・と常時優先度付けをやっても良いと思います。
- 見積もりが先か優先度決めが先か、その都度状況で決めれば良いです

©SQUARE-ENIX 2011

計画の手順

① タスクを洗い出す
   A) ユーザーストーリーの列挙
   B) フィーチャーに分解
   C) タスクに分解
   – LOD式タスク分解
② 見積もる
   – ２点見積もり
   – 見積もりポーカー
   – LOD式見積もり
③ 優先度を決める
④ マイルストーンを定める

©SQUARE-ENIX 2011

マイルストーンを定める

- 詳細な見積もりまで終わり、優先度も決まった状態であれば中長期計画が出来上がったと言えます
- そこから主要なフィーチャーやユーザーストーリーが出来上がる時期を特定し、マイルストーン表を作って分かりやすい目標として表現します
- マイルストーンは「戦略フェイズ」などを行っている初期段階でもラフな状態で作っておくのが良いですが、改めて精度の高いマイルストーンとして調整します

©SQUARE-ENIX 2011

計画のまとめ

① タスクを洗い出す
   A) ユーザーストーリーの列挙
   B) フィーチャーに分解
   C) タスクに分解
   – LOD式タスク分解
② 見積もる
   – ２点見積もり
   – 見積もりポーカー
   – LOD式見積もり
③ 優先度を決める
   – スケジュールの完成
④ マイルストーンを定める

©SQUARE-ENIX 2011

計画を終えて

- この時点でプロジェクトがスケジュール的に破綻しているかどうかはほぼ分かります
- 通常はどうにも破綻が現実の物となって初めて発覚しますから、なにも作業をしていないのに破綻が見つかった場合はむしろ喜ばしい事です
- 改めて開発戦略を見直しましょう

©SQUARE-ENIX 2011

プロジェクトマネジメントの手順

1. **調査する**
2. **戦略を立てる**（リスク一覧/開発戦略マトリクス/価値空間）
3. **設計する**
4. **計画する**（中長期計画）
   - ① タスクを洗い出す（ユーザーストーリー/フィーチャー/タスク/ＬＯＤ式タスク分解）
   - ② 見積もる（２点見積もり/見積もりポーカー/ＬＯＤ式見積もり）
   - ③ 優先度を決める
   - ④ マイルストーンを定める
5. **スプリント**（４週間単位の制作イテレーション）
   - – 階層化ＰＤＣ
   - ① スプリント計画（成果物目標/スプリント計画会/タスク管理シート）
   - ② 日々の制作（朝会/タスク管理ボード）
   - ③ 週の振り返り（週報/週定例）
   - ④ スプリントの振り返り（スプリント報/スプリント振り返り会/成果物発表会）
   - ⑤ 対策・再計画
     - 戦略・設計の修正（ゲーム仕様、アート、プログラム設計などの修正）
     - 中長期計画の修正（タスクの追加・変更、見積もりの変更、優先度の変更）
     - リソースの修正（人員の追加や担当変更など）
   - ⑥ クライアントや上司への報告

©SQUARE-ENIX 2011

スプリントとは

- ２週間あるいは４週間の固定期間（タイムボックス）で計画⇒実行⇒振り返りのイテレーションサイクルを回す単位です
- 私達のプロジェクトでは４週間で運用しています
  - 必ず月曜日始まり、金曜日終わり
  - 祝日などもあるのでだいたい１８～２０営業日
- スクラムというアジャイル系手法で使われている言葉を拝借しましたが内容はかなりアレンジされていますのでスクラムのそれとは別物と捉えてください
- スプリント初日にその４週間分のタスクを中長期のタスクリスト（見積もり済み）から持って来ます
- タスク分解や見積もりの粒度が荒い場合は各スプリントの計画時に詳細化します
- スプリント最終日にはデータ分析などを行って振り返ります

©SQUARE-ENIX 2011

スプリントの構造

©SQUARE-ENIX 2011

再びPDC登場

©SQUARE-ENIX 2011

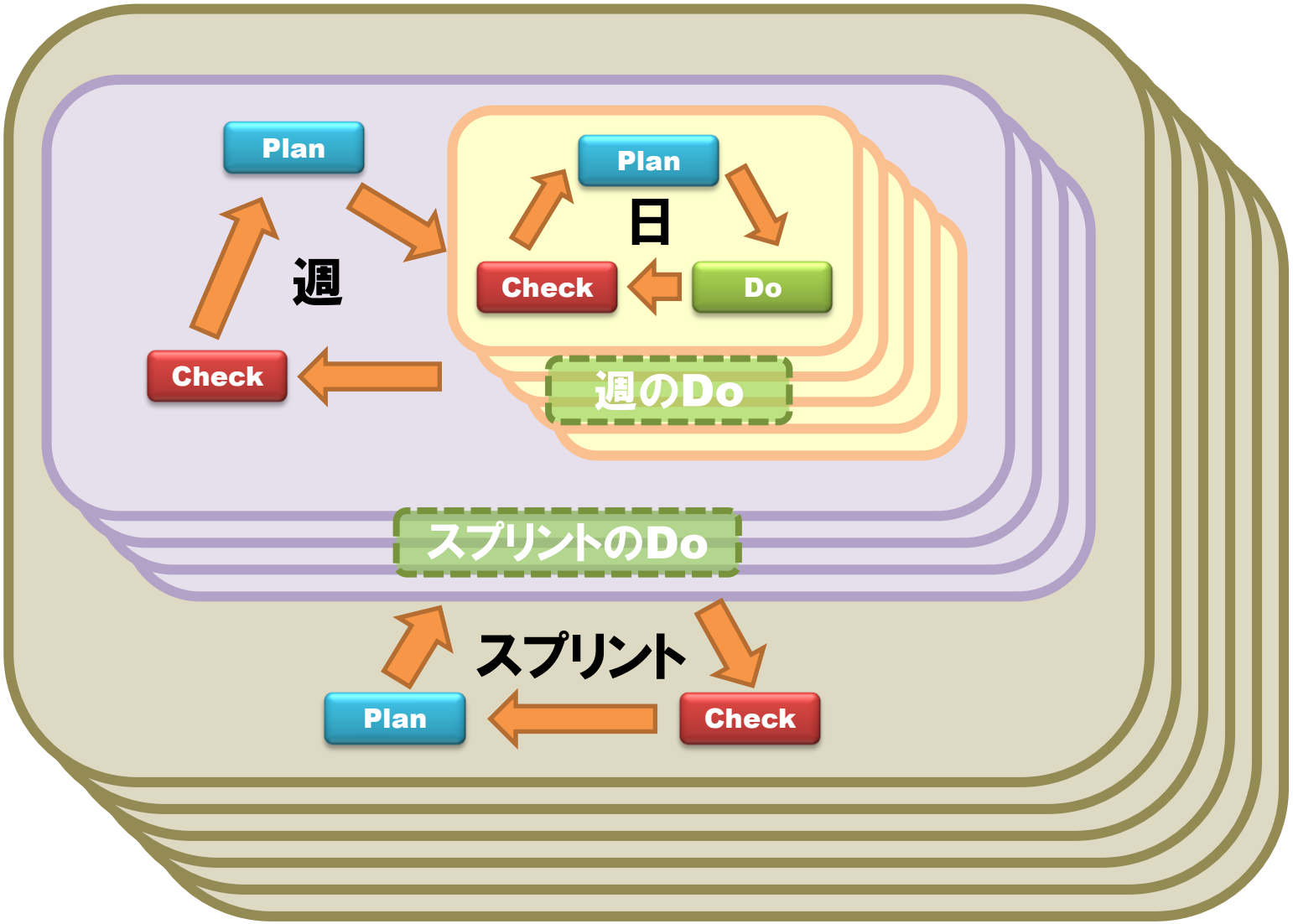
Plan-Do-Checkサイクル

©SQUARE-ENIX 2011

本手法のスプリントでは
PDCを階層化して運用します

©SQUARE-ENIX 2011

階層化されたPlan-Do-Check



©SQUARE-ENIX 2011

階層化PDC

- 各階層でPDCのイテレーションを回します
  - 毎日の単位でPDC
  - 毎週の単位でPDC（Dの中身は毎日のPDC）
  - 毎スプリント（４週間）の単位でPDC（Dの中身は毎週のPDC）
- 階層化を進めることで、大きなサイクル、小さなサイクルいろいろな粒度での問題や計画からの乖離を早い段階で気付き、早い段階で計画の修正をかける事ができます

©SQUARE-ENIX 2011

スプリントの手順

- 階層化PDC

① スプリント計画

② 日々の制作

③ 週の振り返り

④ スプリントの振り返り

⑤ 対策・再計画

⑥ クライアントや上司への報告

©SQUARE-ENIX 2011

スプリントの手順

- 階層化PDC

① スプリント計画
  - 成果物目標
  - スプリント計画会
  - タスク管理シート

② 日々の制作

③ 週の振り返り

④ スプリントの振り返り

⑤ 対策・再計画

⑥ クライアントや上司への報告

©SQUARE-ENIX 2011

スプリント計画

- スプリントの初日にそのスプリントの計画を立てます
- 丸１日スプリント計画で消費しても構いません
- 手順
  1. 成果物目標リストを整備します
  2. 中長期計画からスプリントでこなせそうな分だけのタスクを持って来ます
  3. タスクの粒度が荒い場合は分解を行い、見積もりも分解後のタスクに対して行います
  4. スプリントに入る分のタスクを判定し、成果物目標リストを調整します

©SQUARE-ENIX 2011

成果物目標

- そのスプリントで達成したいユーザーストーリーを“成果物”と見なします
- ユーザーストーリーなので「〇〇が△△できる」という文章で表現します
- スプリントの終わりにこの成果物目標がどれだけ達成出来たかどうかを振り返ります
- 個人の成果物目標を束ねて班の成果物目標とします

©SQUARE-ENIX 2011

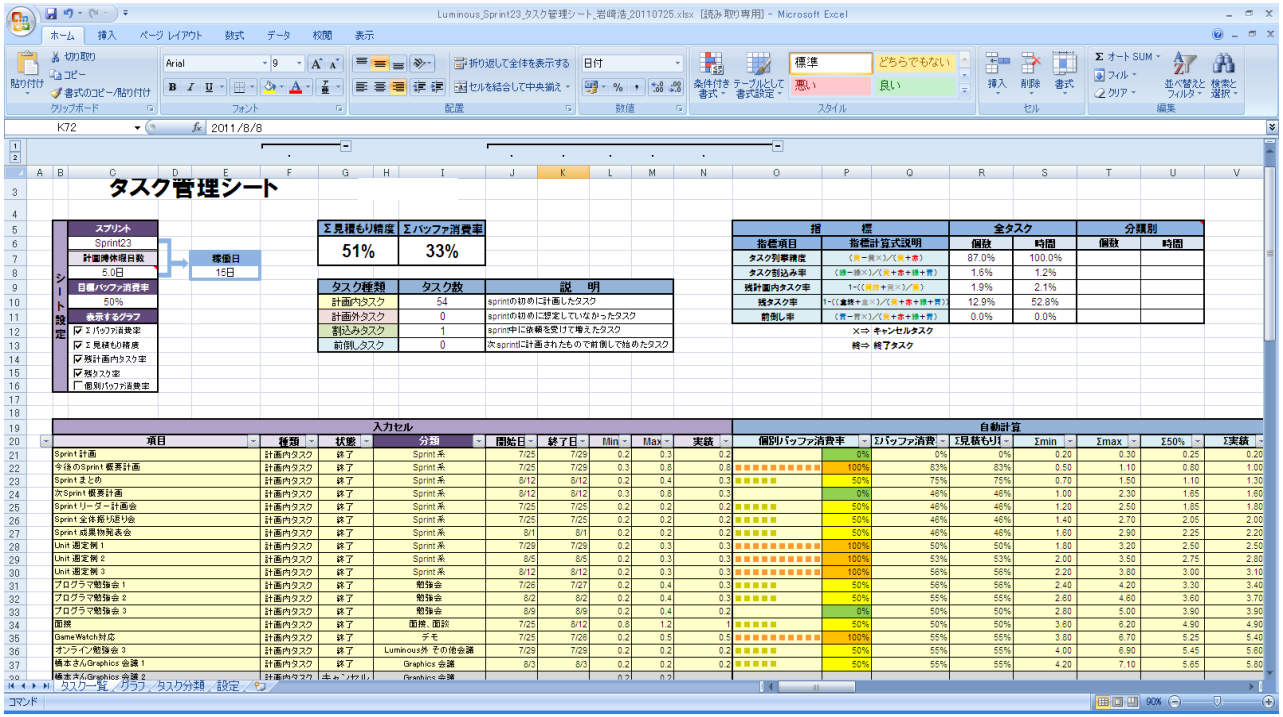
スプリント計画会

- ディレクターと各班の班長が集まって各班の成果物目標を提示し合って、中長期視点も考慮に入れてその成果物目標の設定で問題がないか確認し合う会議です
- スプリント計画会で班の成果物目標が問題無いことが確認された後、各班長はそれを持ち帰り、班の各メンバーのタスク管理シートを整備します

©SQUARE-ENIX 2011

タスク管理シート

- エクセル表で各スプリントのタスクを管理します
- 見積もり精度、バッファー消費率、タスク列挙精度、他、様々な指標が算出されます



スプリントの手順

- 階層化PDC

① スプリント計画
  - 成果物目標
  - スプリント計画会
  - タスク管理シート

② 日々の制作

③ 週の振り返り

④ スプリントの振り返り

⑤ 対策・再計画

⑥ クライアントや上司への報告

©SQUARE-ENIX 2011

スプリントの手順

- 階層化PDC

① スプリント計画
  - 成果物目標
  - スプリント計画会
  - タスク管理シート

② 日々の制作
  - 朝会
  - タスク管理ボード

③ 週の振り返り

④ スプリントの振り返り

⑤ 対策・再計画

⑥ クライアントや上司への報告

©SQUARE-ENIX 2011

日々の制作

- スプリント計画が終わった後は制作作業に入ります
- 「タスク管理ボード」の前で「朝会」を行って日々のPDCサイクルを回します

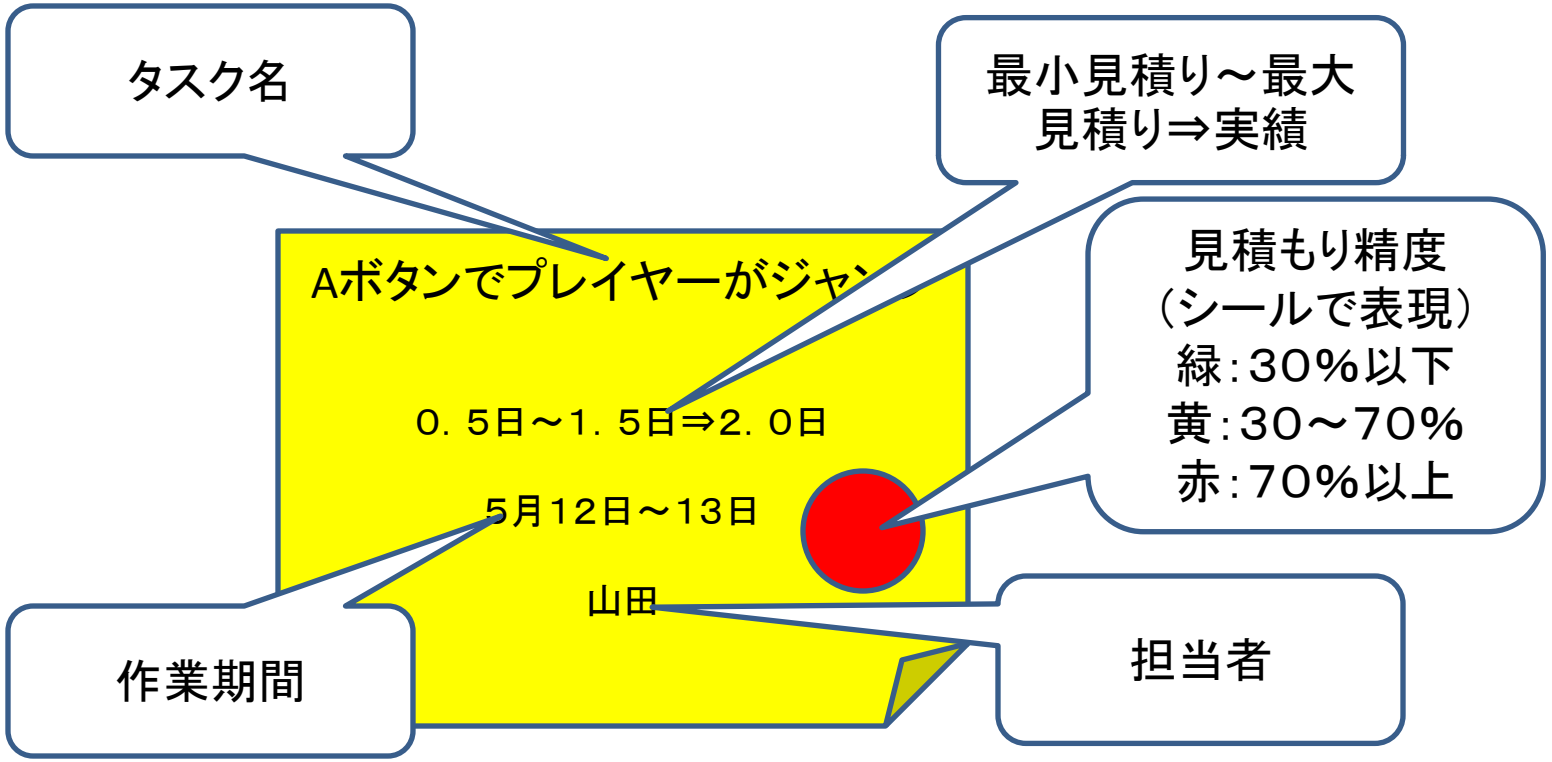
©SQUARE-ENIX 2011

タスク管理ボード

- 朝会を行う場所です
- スプリント期間中の全員分のタスクの進捗状況が管理されます（プロジェクトの人数が多い場合は複数枚用意します）
- 構成
  - そのスプリントで行うタスクを班別に貼る欄
  - その週で行うタスクを班別に貼る欄
  - 作業中のタスクを貼る欄
  - 完了したタスクを個人別に貼る欄

©SQUARE-ENIX 2011

付箋の使い方



- 付箋の色で意味を変えています
  - 黄　スプリント計画時から存在するタスク（計画内タスク）
  - 赤　タスクの分解がうまく行っておらず、スプリント中に増えてしまったタスク（計画外タスク）
  - 緑　人からの依頼で増えたタスク（割り込みタスク）
  - 青　時間が余ったので前倒しで行ったタスク（前倒しタスク）
- 付箋の色と付箋に貼るシールの色によって、タスク管理ボードの完了欄を見れば各自の状況が一目瞭然となります

©SQUARE-ENIX 2011

朝会

- タスク管理ボードの前で行います
- 朝会の班あたり５～８名を目安にしています
- 人数が増えたら班を分割します
- 朝会の班を「制作ユニット別」で組むか「スプリント毎にランダム」で組むかはプロジェクトの特性や目的に応じて使い分けます
- 司会担当者が以下を仕切ります
  - 前日行った作業について問う
    - 作業が終わっていたら実績日数を記入、見積もり精度のシールを貼って、担当者別の完了欄に付箋を移動
  - 本日の作業について問う
    - 新規作業の場合は未着手の付箋を作業中の欄に移動
    - 作業継続の場合は「残り何日」かかるかを宣言
  - 問題や課題がないか問う
- 一人あたり１～３分で、朝会の班あたり１０～１５分で終わらせます
- ひとつのタスク管理ボードで数十名、８班程度まで管理しており、各班が順番に朝会を行います
- 月曜日はその週に行うタスクを「スプリント」の欄から「週」の欄に移動し、「週の計画」とします

©SQUARE-ENIX 2011

スプリントの手順

- 階層化PDC

① スプリント計画
  - 成果物目標
  - スプリント計画会
  - タスク管理シート

② 日々の制作
  - 朝会
  - タスク管理ボード

③ 週の振り返り

④ スプリントの振り返り

⑤ 対策・再計画

⑥ クライアントや上司への報告

©SQUARE-ENIX 2011

スプリントの手順

- 階層化PDC

① スプリント計画
  - 成果物目標
  - スプリント計画会
  - タスク管理シート

② 日々の制作
  - 朝会
  - タスク管理ボード

③ 週の振り返り
  - 週報
  - 週定例

④ スプリントの振り返り

⑤ 対策・再計画

⑥ クライアントや上司への報告

©SQUARE-ENIX 2011

週の振り返り

- 以下の二つを通して週の進捗を振り返ります
  - 週報
  - 週定例

©SQUARE-ENIX 2011

週報

- 各自にメールで全員宛で提出してもらいます
- 「個人週報」と「制作ユニット週報」があります（右は個人週報のフォーマット）

==========
週次作業報告書（個人）
==========

【報告者】
○○　○○

【作業期間】
２０１１年○月○日（月）～○月○日（金）

【バッファ消費率】
○○％

【作業内容】

【成果物】

【計画からの差異】

【良かった点】

【問題点⇒改善策】

【翌週の計画】

【自由コメント】

©SQUARE-ENIX 2011

週定例

- 各制作ユニットのリーダーがユニットのメンバーを招集して、毎週金曜日に担当する班の進捗を確認します
- 事前に個人週報を提出してもらって、それを元に状況を分析し、必要に応じて対策を講じます
- その結果は班長からチーム全体に「制作ユニット週報」としてメールで送られます
- ディレクターと各ユニットのリーダーが集まった「全体週定例」を別に行うケースもあります（プロジェクトの特性に応じて方針を変えています）

©SQUARE-ENIX 2011

スプリントの手順

- 階層化PDC

① スプリント計画
  - 成果物目標
  - スプリント計画会
  - タスク管理シート

② 日々の制作
  - 朝会
  - タスク管理ボード

③ 週の振り返り
  - 週報
  - 週定例

④ スプリントの振り返り

⑤ 対策・再計画

⑥ クライアントや上司への報告

©SQUARE-ENIX 2011

スプリントの手順

- 階層化PDC

① スプリント計画
  - 成果物目標
  - スプリント計画会
  - タスク管理シート

② 日々の制作
  - 朝会
  - タスク管理ボード

③ 週の振り返り
  - 週報
  - 週定例

④ スプリントの振り返り
  - スプリント報
  - スプリント振り返り会
  - 成果物発表会

⑤ 対策・再計画

⑥ クライアントや上司への報告

©SQUARE-ENIX 2011

スプリントの振り返り

- ４週間のスプリントが終わるとそのスプリントの進捗具合や問題点を振り返ります
- 「スプリント報」と「スプリント振り返り会」が振り返りの中心となります
- スプリント振り返り会は全員で行います
- それぞれ、週報や週定例と比べてより緻密な分析を行います
- 今回は詳細は省略します

©SQUARE-ENIX 2011

数値見積りミスと作業の列挙漏れ

- 振り返りの際には計画の変動要因として主に二つの要因を意識しましょう
  - 数値見積りミス
  - 作業の列挙漏れ

©SQUARE-ENIX 2011

成果物発表会

- 「スプリントの振り返り会」に連結して「成果物発表会」も行っています
- 各ユニットで目に見える成果物がある場合は皆の前で実際に操作してプレゼンテーションします
- 見せる側も、見る側もモチベーションが上がりますし、チーム内の意識統一もしやすくなるので、プロジェクトにとって重要な場となっています
- 見せる側は「成果物発表会までに良い感じに仕上げる！」という気合が入ってくれるプラス効果もあります

©SQUARE-ENIX 2011

スプリントの手順

- 階層化PDC

① スプリント計画
  - 成果物目標
  - スプリント計画会
  - タスク管理シート

② 日々の制作
  - 朝会
  - タスク管理ボード

③ 週の振り返り
  - 週報
  - 週定例

④ スプリントの振り返り
  - スプリント報
  - スプリント振り返り会
  - 成果物発表会

⑤ 対策・再計画

⑥ クライアントや上司への報告

©SQUARE-ENIX 2011

対策・再計画

- スプリントの進捗を振り返って、必要に応じて対策や再計画を行います
- 対策・再計画例
  - 作業優先度の見直し
  - 担当者の配置換え
  - 設計の見直し
  - 人員採用計画の修正

©SQUARE-ENIX 2011

スプリントの手順

- 階層化PDC

① スプリント計画
  - 成果物目標
  - スプリント計画会
  - タスク管理シート

② 日々の制作
  - 朝会
  - タスク管理ボード

③ 週の振り返り
  - 週報
  - 週定例

④ スプリントの振り返り
  - スプリント報
  - スプリント振り返り会
  - 成果物発表会

⑤ 対策・再計画

⑥ クライアントや上司への報告

©SQUARE-ENIX 2011

クライアントや上司への報告

- スプリントが終わる度にクライアントや上司へ進捗を報告しましょう
- 進捗報告時に伝えること
  - 旧スプリントの成果物（目に見える成果物を中心に）
  - 新スプリントでの成果物目標の宣言
  - 問題点・課題とその対策案
  - 中長期計画やマイルストーンの変動情報
- クライアントや上司への進捗報告を定期的にしっかり行う事でプロジェクトやプロジェクトリーダーへの信頼感が上昇します
- 問題や課題を隠すのではなく、早期に積極的に開示して、出来ることならば一緒にプロジェクトの未来を切り開く仲間として、クライアントや上司に味方になってもらいましょう

  ※なかなかそううまくは行かない環境も存在するとは思いますが

©SQUARE-ENIX 2011

プロジェクトマネジメントの手順

1. **調査する**
2. **戦略を立てる**（リスク一覧/開発戦略マトリクス/価値空間）
3. **設計する**
4. **計画する**（中長期計画）
   - ① タスクを洗い出す（ユーザーストーリー/フィーチャー/タスク/ＬＯＤ式タスク分解）
   - ② 見積もる（２点見積もり/見積もりポーカー/ＬＯＤ式見積もり）
   - ③ 優先度を決める
   - ④ マイルストーンを定める
5. **スプリント**（４週間単位の制作イテレーション）
   - – 階層化ＰＤＣ
   - ① スプリント計画（成果物目標/スプリント計画会/タスク管理シート）
   - ② 日々の制作（朝会/タスク管理ボード）
   - ③ 週の振り返り（週報/週定例）
   - ④ スプリントの振り返り（スプリント報/スプリント振り返り会/成果物発表会）
   - ⑤ 対策・再計画
     - 戦略・設計の修正（ゲーム仕様、アート、プログラム設計などの修正）
     - 中長期計画の修正（タスクの追加・変更、見積もりの変更、優先度の変更）
     - リソースの修正（人員の追加や担当変更など）
   - ⑥ クライアントや上司への報告

©SQUARE-ENIX 2011

本プロジェクトマネジメントの方法で得られる効果

- **不確実性を制御**
  - 不確実性が減る
  - 見通しが立つ
  - 早期に問題発見可能
  - 計画修正や対策もしやすい
- **メンタル効果**
  - スタッフ自身の作業と期間の見通しが立つので、無根拠になんとなく日々残業しないとならないのかもしれないというプレッシャーから解放される⇒残業や休日出勤が大幅減
  - お互いの仕事を知ることができる
  - 楽しくなる
- **クライアントや上司からの信頼を得る**
- **目標通りに満足行くプロジェクトの成果がしっかり出る**

©SQUARE-ENIX 2011

より詳しく知りたい方へ

©SQUARE-ENIX 2011

推薦書籍

©SQUARE-ENIX 2011

推薦書籍

- **「マンガでわかる プロジェクトマネジメント」**
  - PM世界標準PMBOKのダイジェストをマンガで
- **「アジャイルサムライ」**
  - 読みやすいアジャイル本
- **「最短で達成する 全体最適のプロジェクトマネジメント」**
  - CCPMを知る
- **「XPエクストリーム・プログラミング実行計画」**
  - XPにおける計画を知る
- **「アジャイルソフトウェア開発スクラム」**
  - スクラムを知る
- **「Agile Game Development with Scrum」**
  - ゲーム開発におけるスクラム実践例
- **「アジャイルな見積もりと計画づくり」**
  - 最もお勧めであり、最も考え方の近い本
  - 分厚く内容も高度なので最後に仕上げで読むのが良いです

©SQUARE-ENIX 2011

「デジタルゲームの技術」
第９章でもプロジェクトマネジメントについて
インタビュー形式で語っています

©SQUARE-ENIX 2011

来年プロマネ本が出版されます

- タイトル
  **「スクウェア・エニックスの
  ゲーム開発プロジェクトマネジメント（仮題）」**
- 出版時期
  - ２０１２年６月予定
- 著者
  - 橋本　善久（スクウェア・エニックス CTO）
- 出版社
  - ソフトバンククリエイティブ
- 内容
  - プロジェクトが失敗するメカニズム
  - プロジェクトを成功させる仕組みとメカニズム
  - 詳細なプロジェクトマネジメントマニュアル
  - ゲーム開発固有の課題へ向けて
  - スクウェア・エニックスにおける運用実例集
  - など（今回触れられなかった事を含めて語り尽くします）

©SQUARE-ENIX 2011

プロジェクトマネジメントは
とても奥深く
とても楽しい活動です

©SQUARE-ENIX 2011

やり方によって
開発者やユーザーや会社を
ハッピーにも不幸にもできます

©SQUARE-ENIX 2011

プロジェクトマネジメントを
うまく行うコツは
「そのやり方をなぜ行うのか」
を常に深く考察して徹底的に
心理力学や情報の流れ
を意識して論理的に仕組みを
組み上げ改善し続ける
事にあります

©SQUARE-ENIX 2011

原典主義に陥ってはなりませんし、
かと言ってなにも考えずに
自己流に走ってもなりません

©SQUARE-ENIX 2011

すべてに疑問を持ちましょう
すべてに自分の答えを用意しましょう

- なぜスプリントが必要なのか？なぜタイムボックス化は有効なのか？
- スプリントは１週間、２週間、４週間、それ以上、どれくらいの期間が良いのか？
- スプリントの期間は途中で変更しても良いか？
- なぜ朝会があるのか？夕方の場合はどういう効果の違いがあるか？
- 朝会の一つの班の最少、最大人数は何人であるべきか？
- なぜ２点見積もりなのか？１点や３点ではだめなのか？
- 見積もりの単位は“時間”“日”“週” どれが良いのか？各単位のメリット・デメリットは？
- プロジェクト管理ボードはどういう構成であるべきか？
- なぜ付箋は有効なのか？
- 付箋にはどういった情報を載せるべきか？
- デジタル管理をするべきか？アナログ管理をするべきか？
- プロジェクト管理シートはどういう構成であるべきか？
- 定例会議は行うべきか？行うならばどういうスタイルであるべきか？
- 日報は必要か？週報は必要か？スプリント報は必要か？
- 各報告書はどのようなフォーマットであるべきか？
- そのやり方で誰にどんな情報がいつ届くのか、誰に届かないのか
- そのやり方でどんな心理効果が生まれるのか
- などなどなどなど・・・

©SQUARE-ENIX 2011

「本にこう書いてあったから」
「あの人がああ言っていたから」ではなく、
自分で「なぜ」「どうやって」を深く考察して
納得の上で各“仕組み”を導入してみましょう。
（※やってみないと分からない仕組みの場合は「実験である」という
明確な意識下でお試し導入しましょう。）
そして実行しながら効果を測り、効果を感じ、
“仕組み”を調整し続けましょう。

©SQUARE-ENIX 2011

すべての情報を全力で肯定し
すべての情報を全力で否定し
頭にたくさん汗をかいて
ひとつひとつの“仕組み”に対して
「なぜそうなのか」を
あなたなりに自力で導き出してください

※本講義も「全力の肯定、全力の否定」の対象であるべきことは言うまでもありません

©SQUARE-ENIX 2011

思考停止しない

©SQUARE-ENIX 2011

それがプロジェクトマネジメントを
なるべくうまく行かせるための
最大のポイントなのです

©SQUARE-ENIX 2011

ぜひあなたなりの深い考察をした
プロジェクトマネジメント方法論を
構築してください

©SQUARE-ENIX 2011

機会があったら
それをぜひ聞かせてください

ディスカッションしましょう

©SQUARE-ENIX 2011

ありがとうございました

©SQUARE-ENIX 2011