テスト技術者のための ソフトウェアメトリクス入門

ー信頼性を測定し予測する正しい作法ー

野中 誠（東洋大学）
鷲崎 弘宜（早稲田大学）

JaSST2010

2010年1月28日

『演習で学ぶソフトウェアメトリクスの基礎』

目次

本日の内容は、
この内容を中心に
ご紹介します

日経BP社より好評発売中！

講演概要

1. イントロダクション
2. 欠陥・テストデータによるプロジェクト進捗の把握方法
3. プロセスの欠陥除去能力を評価するメトリクス
4. 動的モデルによる欠陥数予測
5. システムの信頼性と可用性との関わり
6. まとめ

1. イントロダクション

- メトリクスが必要な理由
- メトリクスの限界を知り、意思決定に役立てる
- 欠陥データは貴重な情報源
- 欠陥とは？

なぜメトリクスが必要なのか？ 一般的・教義的な理由

What you cannot measure is neither predict nor control.
測れないものは、予測したり管理したりすることができない（ケルビン卿）

What you measure is what you get.
測定したものが、わかったことだ（キャプラン＆ノートン）

What is “Software Engineering”？
体系的・規律的・定量的なアプローチを、ソフトウェア開発・運用・保守に適用すること、およびその研究（IEEE標準用語集）

CMMI プロセス領域…
- レベル2「管理された」
  → 監視と制御、測定と分析
- レベル4「定量的に管理された」
  → 定量的プロジェクト管理

「ありがたい言葉」もいいが、
もっと、身近なニーズから
「メトリクス」に取り組みたい…

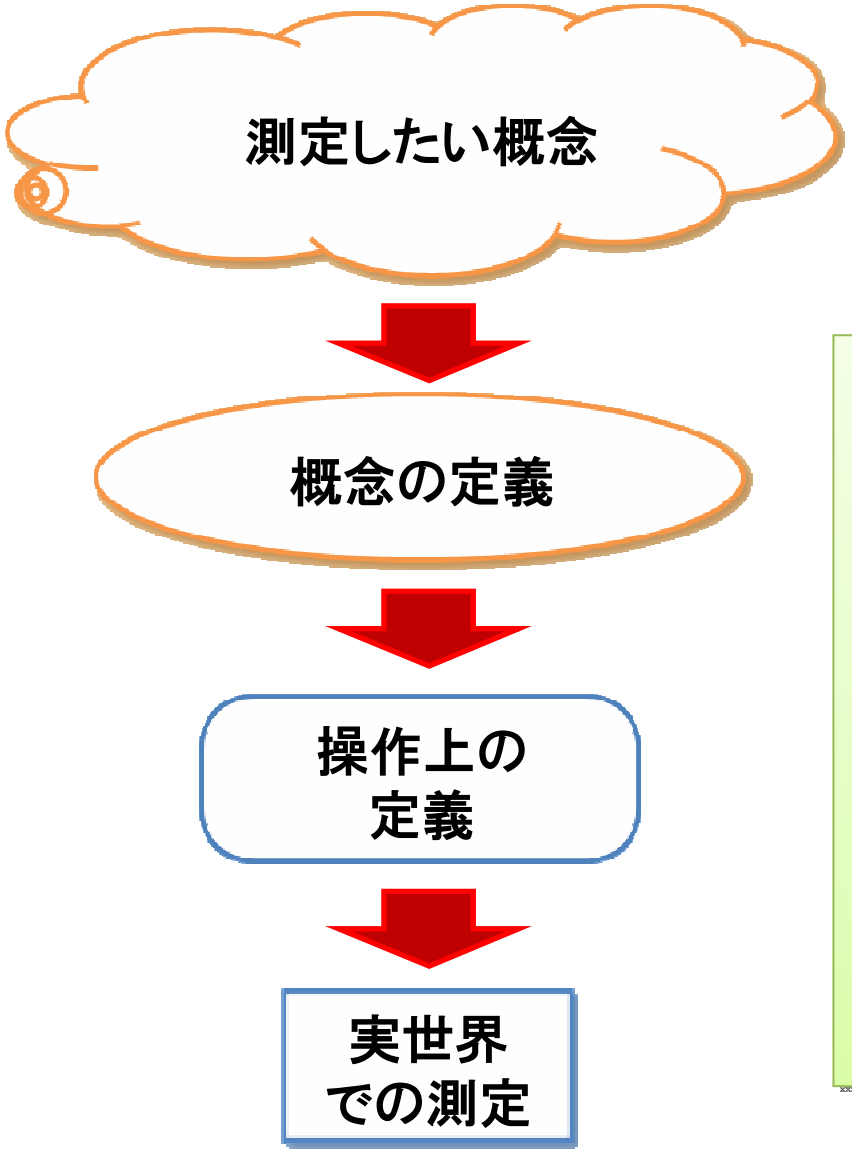
なぜメトリクスが必要なのか？ Informed decision指向

- 「必要な情報」を得た上での意思決定のために
  - ソフトウェア／プロジェクトといった抽象的・複雑な対象について、
  - そのマネジメントにおける様々な意思決定に役立つ情報を、
  - 一貫した方法で獲得し、論理的な判断を下し、リスクを把握し、
  - そして、前へ前へと進んでいきたい
- 「必要な情報」の獲得・伝達手段として
  - 主観的情報も、意思決定の際に用いている　例）説明の論理性を判断
  - 主観的情報は、情報収集の一貫性、効率性、説明性に問題がある
  - メトリクスを定めることで、必要な情報を、効果的に、一貫して収集できる
  - 組織やメンバーに説明可能な根拠情報として利用できる

メトリクスに取り組まない合理的理由は、極めて限定的である

L. M. Laird and M. C. Brennan, *Software Measurement and Estimation: A Practical Approach*, John-Wiley and Sons, 2006.
（野中・鷲崎訳：演習で学ぶソフトウェアメトリクスの基礎、日経BP社(2009)） 訳者前書きより

メトリクスの限界を知り、意思決定に役立てる



- 「実世界で測定」できることは、「測定したい概念」のごく一部に過ぎない
- 過度の期待を抱いてはいけない
- メトリクスの限界を理解した上で、“informed decision”に役立てる

L. M. Laird and M. C. Brennan, *Software Measurement and Estimation: A Practical Approach*, John-Wiley and Sons, 2006.
(野中・鷲崎訳:演習で学ぶソフトウェアメトリクスの基礎、日経BP社(2009))

欠陥データは、貴重な情報源

- 一般的・常識的な視点
  -  欠陥は **望ましくない**
  -  できるだけ **早く除去したい**
- メトリクス技法を身につけたソフトウェア技術者の視点

  （上記を共有するのは大前提として…）

  欠陥データは、さまざまな観点に役立つ貴重な情報源である
  -  **スケジュール**：プロジェクトの進捗を把握できる
  -  **プロセスエンジニアリング**：プロセスの有効性を評価できる
  -  **品質**：プロダクトの欠陥密度を早期に予測できる
- 欠陥データの特性
  - 識別できる
  - 測定できる
  - 予測できる
  - 傾向を把握できる

欠陥データは、
ソフトウェア開発で利用可能なデータのうち、
最良で、もっとも役立つデータのひとつである

欠陥に関連する用語

IEEE標準、JIS X 0014などに見る「欠陥」に関連する用語

- エラー・誤差・誤り(error)
  - 計算、観測若しくは測定された値または状態と、真の、指定された若しくは理論的に正しい値または状態との間の相違
  - 狭義にはmistakeを意味する場合もある
- ミス(mistake)
  - 不正確な結果を作り出す人間の行為
- 障害・フォールト(fault)
  - 要求された機能を遂行する機能単位の能力の、縮退または喪失を引き起こす、異常な状態
  - プログラム中の不正確なステップ、プロセス、またはデータ定義
- 故障(failure)
  - 要求された機能を遂行する、機能単位の能力がなくなること

欠陥(defect)は？？　→　文献によって定義が異なる

「欠陥」に含まれるもの

- 障害(fault)
  - 故障(failure)の原因部分
  - ソフトウェアの振る舞いに問題を引き起こす原因部分
  - 不正確なステップ、プロセス、データ定義 (IEEE標準用語集)

- 仕様や標準とのズレ(anomaly)
  - 仕様とのズレ、規約違反など、標準との不適合箇所も「欠陥」と見なす

これらの両方を「欠陥」として記録し、プロジェクトを通して管理対象とする

Mäntylä, M. V. and C. Lassenius (2009). "What Types of Defects Are Really Discovered in Code Reviews?" *IEEE Trans. Software Eng.*, 35(3): 430-448.
IEEE Standard Glossary of Software Engineering Terminology, IEEE (2002)

ソフトウェア測定に関する基本用語

- メトリクス (metric / metrics)
  - a quantitative measure of the degree to which a system, component, or process possesses a given attribute (IEEE Std 610.12-1990)
  - 定義された測定方法及び測定尺度 (ISO/IEC 14598-1: 1998)

- 測定量 (measure)
  - 測定の結果として値が割り当てられる変数 (ISO/IEC 15939:2002)

- 尺度 (scale)
  - 連続的もしくは離散的な値の順序集合、又は分類の集合で、それに属性を対応付けるもの (ISO/IEC 15939:2002)

2. 欠陥とテストのデータを用いて プロジェクトの進捗を把握する

- 成功テストケース数の追跡グラフ
- テスト管理図
- 修正未了欠陥(欠陥バックログ)の推移
- コード結合計画と欠陥発見計画

プロジェクト進捗把握に有効なメトリクス

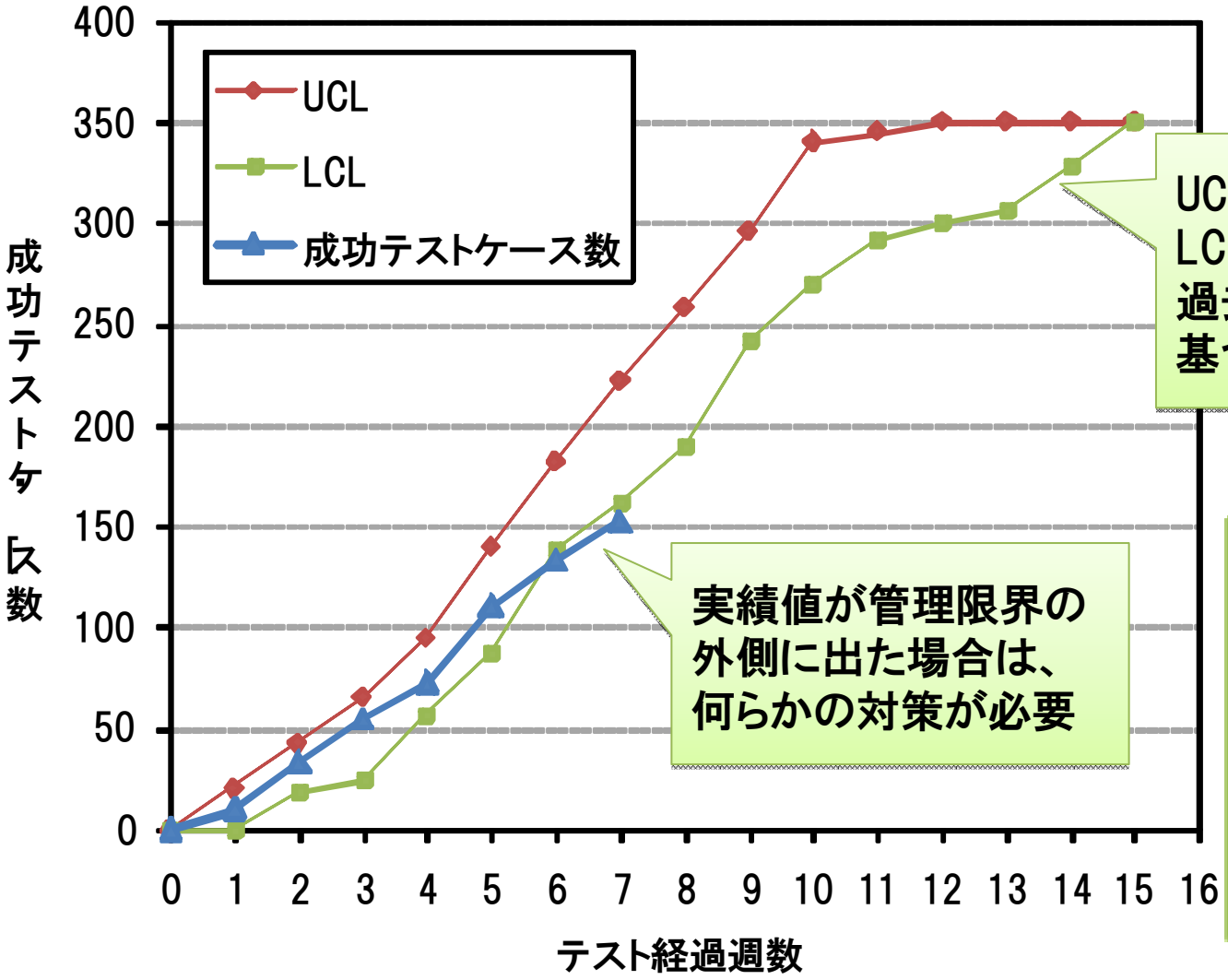
- マイルストーン
  - SMART (Specific, Measureable, Achievable, Relevant, Time based) にする
- コード結合
  - コード作成～単体テスト～結合作業を対象
  - 「コード結合メトリクス」「コード結合パターン」により状況を把握

この内容を紹介

- テストの進捗
  - 正式テストを対象
  - 「実行済みテストケース数」「成功テストケース数」により把握
- 欠陥の発見・修正
  - 「欠陥発見パターン」を用いて、リソース割当計画に活用
  - 例)コード結合計画に欠陥発見計画を重ね合わせてバックログを予測
- プロセス有効性
  - 計画通りに進んでいるか、過去実績に対して妥当かどうかを判断する

テストの進捗：成功テストケース数の追跡グラフ

時系列の成功テストケース数をグラフに表し、テストの進捗を把握する



UCL（上方管理限界）と
LCL（下方管理限界）を、
過去の類似プロジェクトに
基づいて設定している

実績値が管理限界の
外側に出た場合は、
何らかの対策が必要

「実行済テストケース数」
を重ね合わせてもよい

↓

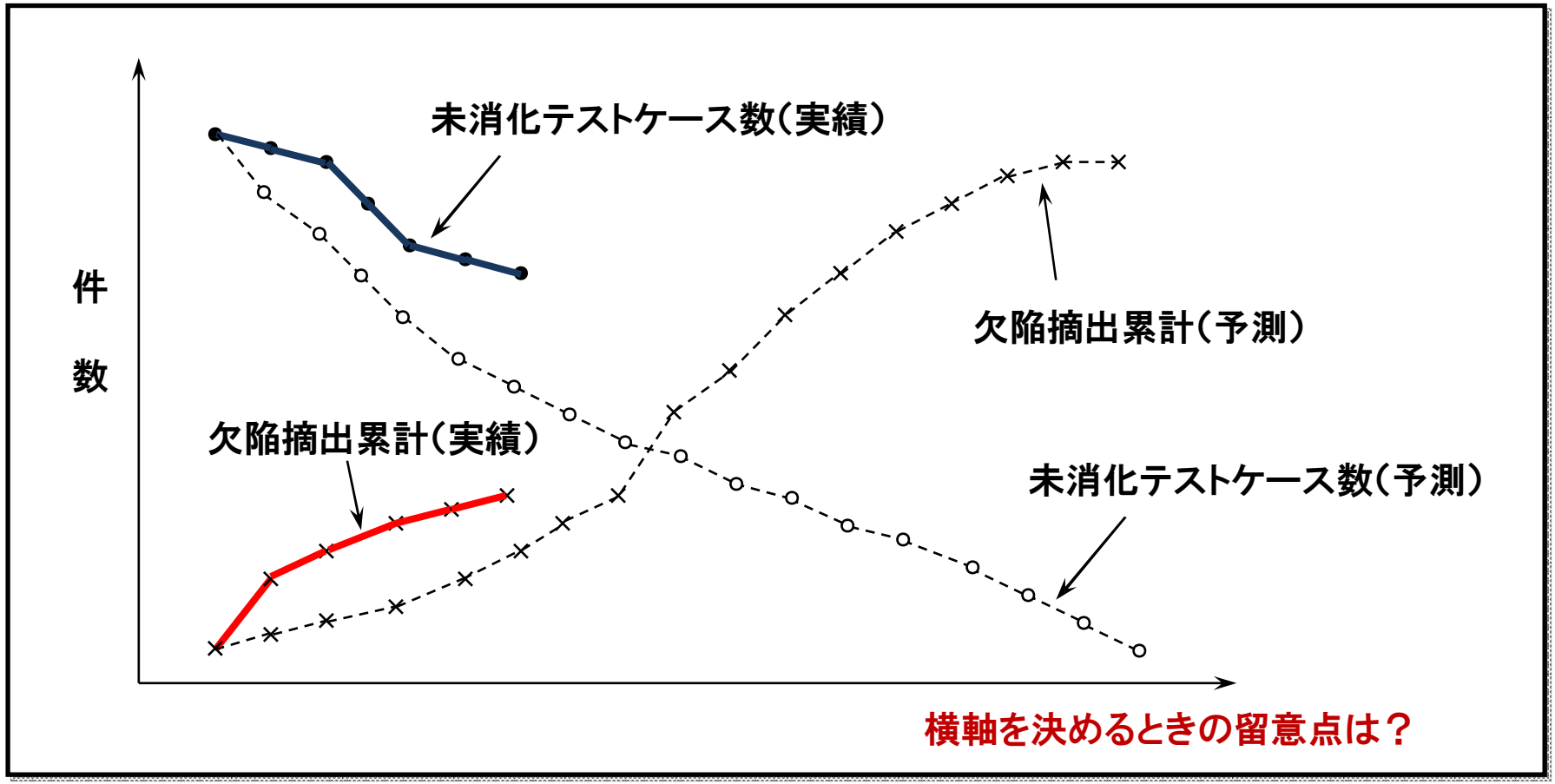
・テストケース実施率
・成功テストケース率
などを把握できる

テストの進捗:テスト管理図

次の4つの情報をグラフで表す

① 欠陥摘出累計(予測)　② 欠陥摘出累計(実績)

③ 未消化テストケース数(予測)　④ 未消化テストケース数(実績)



テストの進捗：テスト管理図の横軸

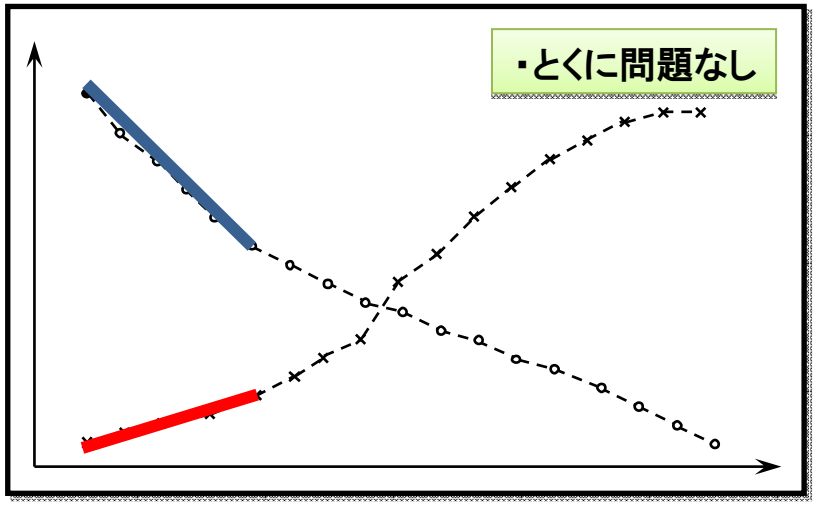
次の状況を考えたときに、テスト管理図の横軸をどう設定するか？

週	1	2	3	4
欠陥摘出数	10	5	3	2
テスト時間	80	40	40	20
週あたりの欠陥摘出数	10	5	3	2
テスト時間あたりの欠陥摘出数	0.13	0.13	0.08	0.10

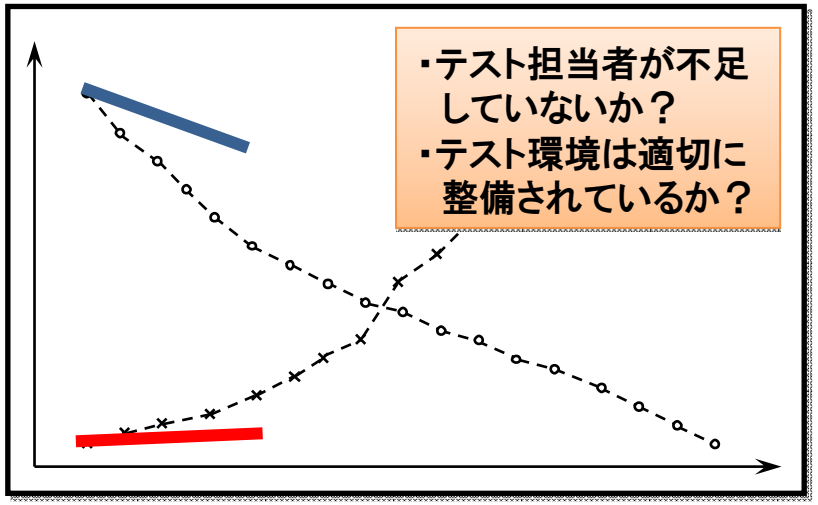
テスト管理図の横軸には、テスト日数ではなく、テスト時間をとる方が望ましい
テスト日数をとる場合は、テスト時間のばらつきを考慮する必要がある

テストの進捗：テスト管理図のパターン

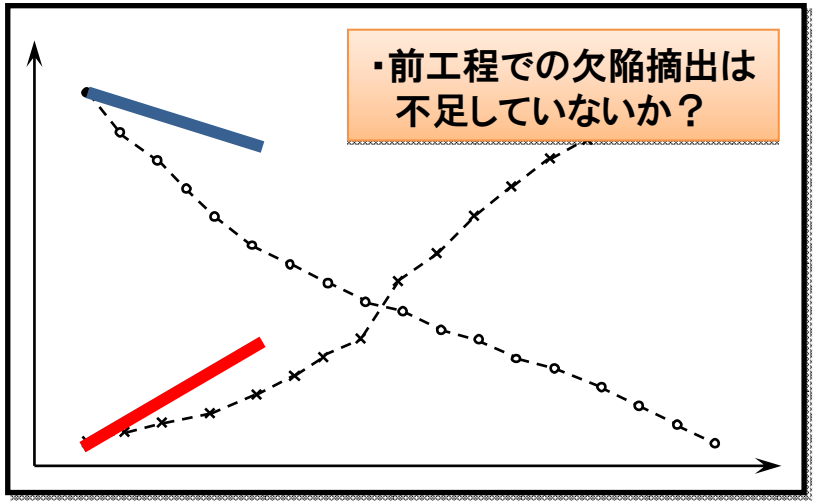
計画通りに進行



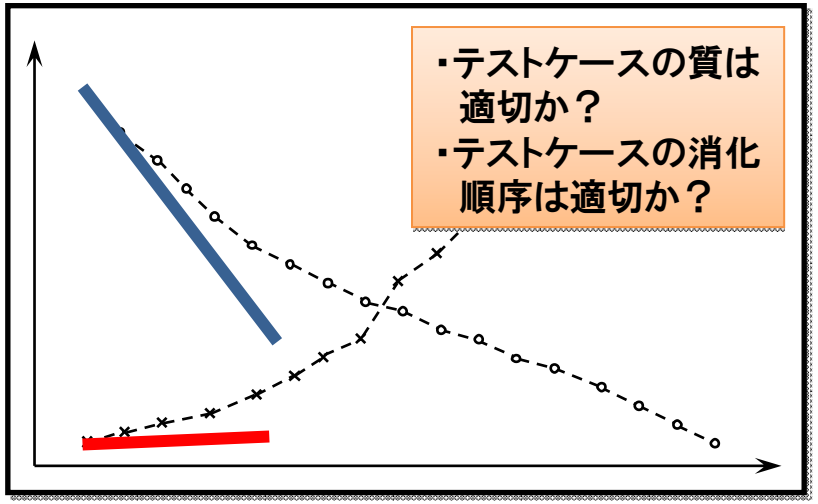
テストケース消化停滞・欠陥摘出不足



テストケース消化停滞・欠陥多発

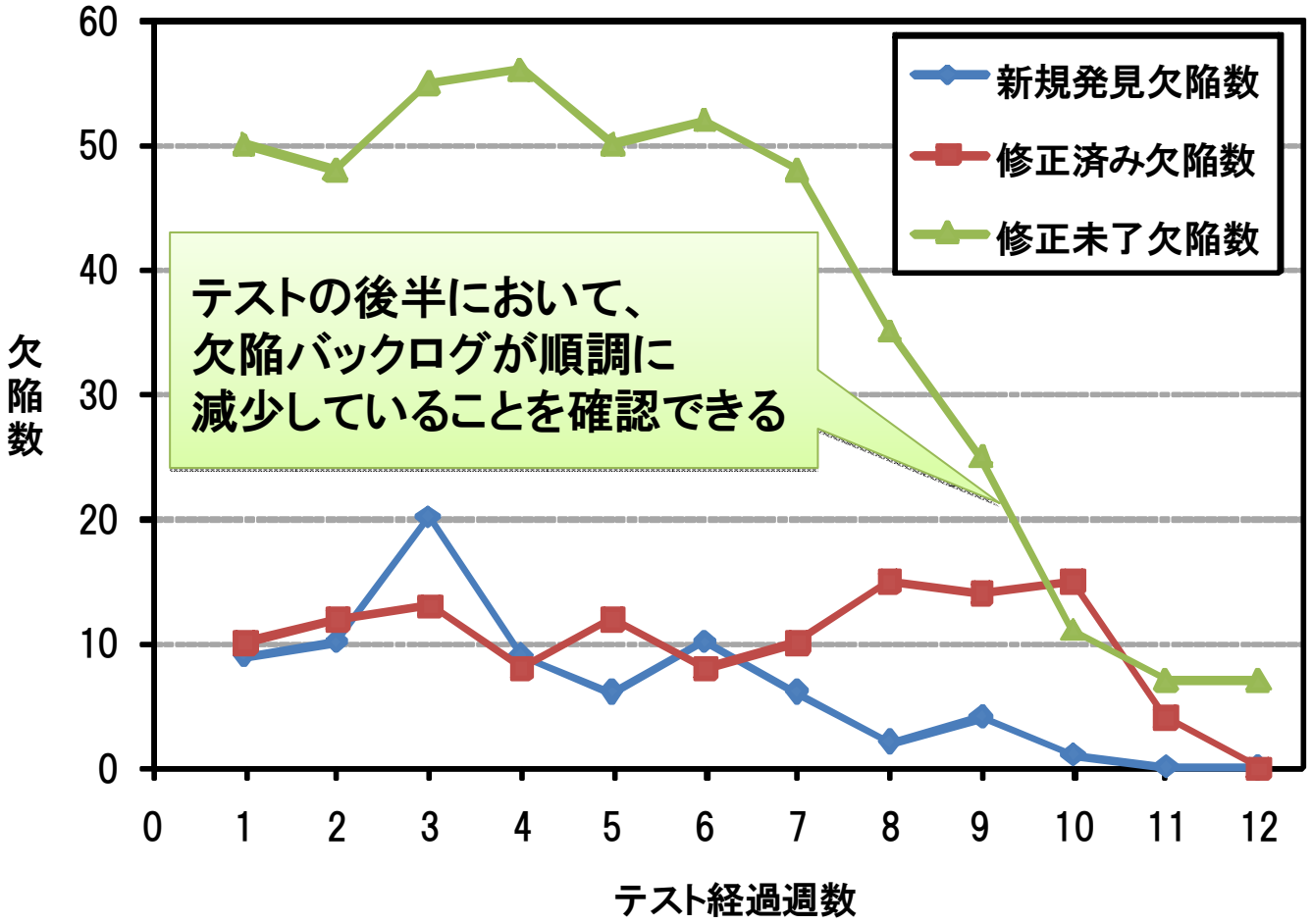


テストケース急速消化・欠陥摘出不足



欠陥の発見と修正：修正未了欠陥の推移

テスト経過週数に対する欠陥発見数／修正数を追跡し、
修正未了欠陥（欠陥バックログ）の推移を把握する



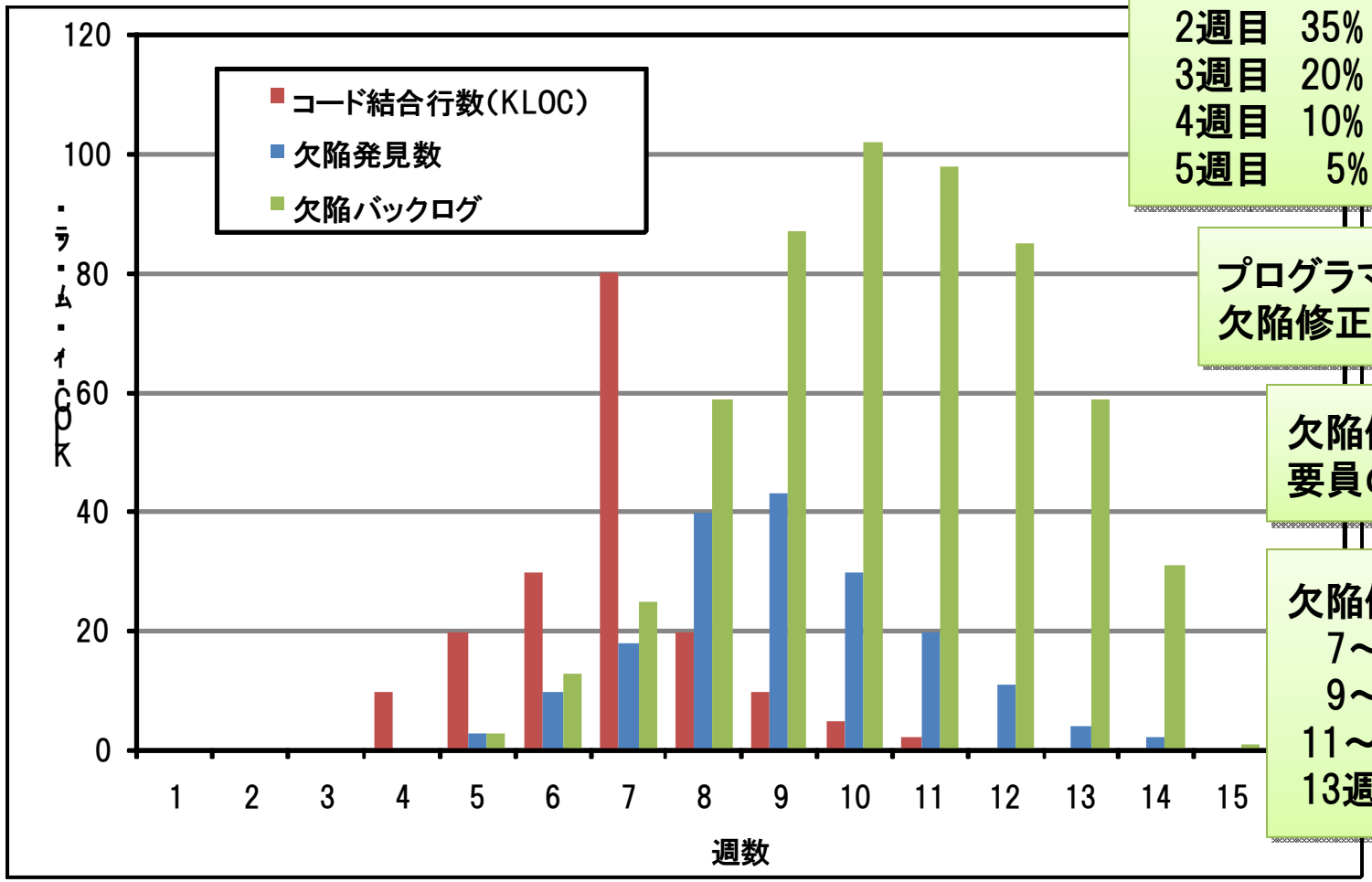
どのようにリソースを
割り当てたときに、
どのようなパターンで
欠陥バックログが解消
したのかを把握する

↓

欠陥発見計画に利用

演習1:コード結合計画と欠陥発見計画

欠陥発見計画の立案と、欠陥バックログの予測を行ってみましょう



欠陥作り込み率:1欠陥/KLOC

コード結合後の欠陥発見パターン
- 1週目 30%
- 2週目 35%
- 3週目 20%
- 4週目 10%
- 5週目 5%

プログラマー1人あたりの欠陥修正能力:5欠陥/週・人

欠陥修正を担当できる要員の人数:6人/週

欠陥修正の要員工数率
- 7～ 8週: 20%
- 9～10週: 50%
- 11～12週: 80%
- 13週～ :100%

演習1:コード結合計画と欠陥発見計画

週	コード結合 (KLOC)	欠陥修正要員(人)	要員時間配分比率	当該週結合コードの欠陥発見数 4	5	6	7	8	9	10	11	欠陥発見 週別	累積	欠陥修正 週別	累積	欠陥バックログ
1	0	0	0%									0	0	0		
2	0	0	0%									0	0	0		
3	0	0	0%									0	0	0		
4	10	0	0%									0	0	0	0	0
5	20	6	0%											0	0	
6	30	6	0%											0	0	
7	80	6	20%											6	6	
8	20	6	20%				24					40	71	6	12	59
9	10	6	50%				28	6				43	114	15	27	87
10	5	6	50%				16	7	3			30	144	15	42	102
11	2	6	80%				8	4	4	2		20	164	24	66	98
12	0	6	80%				4	2	2	2	1	11	175	24	90	85
13	0							1	1	1	1	4	179	30	120	59
14	0								1	1		2	181	30	150	31
15	0											0	181	30	180	1
16	0	6	100%									0	181	30	210	0
合計	177						80	20	11	6	2					

第4週に結合した10KLOCに含まれる10個の欠陥が、第5週目以降から発見される

第5週結合の20KLOC

欠陥バックログが発生する

第4週に結合した10KLOCには1欠陥/KLOC * 10KLOC =10個の欠陥が含まれると予想できる

ここでのまとめ

- ソフトウェアプロジェクトには、リスクがつきもの
- プロジェクトの進捗を適切に把握することが重要
- 進捗把握のメトリクスには、次を考えよう
  - “SMART” なマイルストーンを定義する
  - コード結合メトリクス、コード結合パターンを把握する
  - 実行済み／成功テストケース数を測定する
  - 欠陥発見／修正パターンを把握し、バックログを監視する
  - すべての主要工程に、プロセス有効性メトリクスを定める

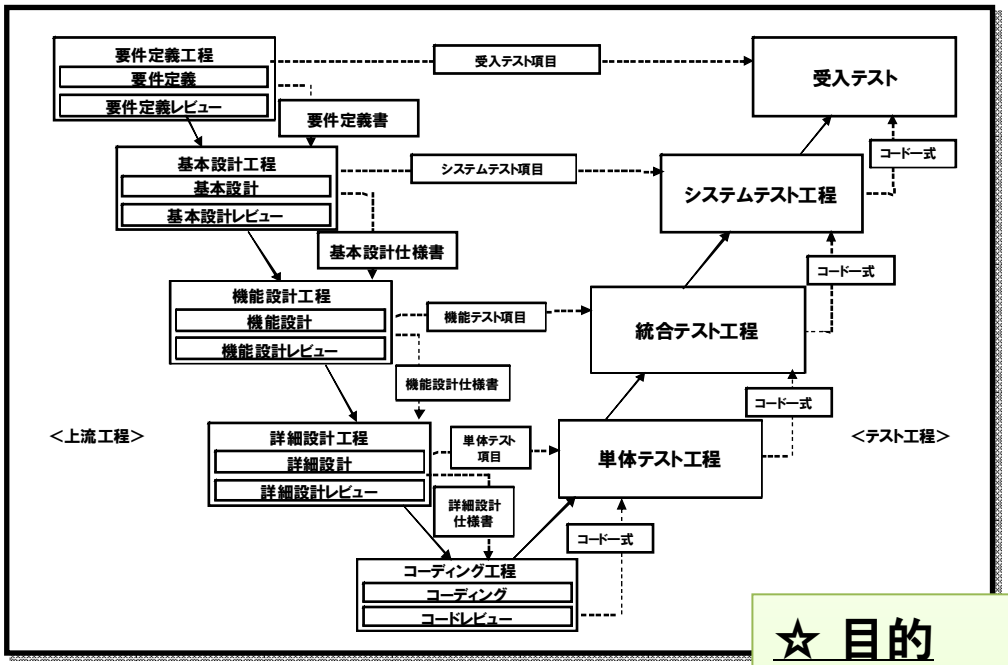
- 実績を追跡し、目標と比較できるようになろう
- その結果に基づき、状況に応じて是正措置がとれるようになろう

3. プロセスの欠陥除去能力を評価するメトリクス

- 出荷後の欠陥密度
- 欠陥除去率

出荷後の欠陥密度

ソフトウェア開発プロセス



製品出荷

評価タイミング

アウトプットである ソフトウェア製品の測定項目

- 規模
  - ー 新規・変更KLOC
- 品質
  - ー 出荷後に見つかった欠陥数
    （顧客指摘／保守チーム指摘）
    ※通常は出荷後1年で区切る

出荷後の欠陥密度

$$\frac{\text{出荷後1年間に見つかった欠陥の数}}{\text{新規・変更KLOC}}$$

☆ 目的

- まずは、最終成果物の品質を把握する
- 開発プロセス全体としてのパフォーマンスを把握する

☆ POINT

- 規模と品質の定義は、製品や組織内で統一しなければ一貫性がなくなってしまう
- 製品分野、利用者数、稼働時間数などの違いを考慮した方がよい

出荷後の欠陥密度のベンチマークデータ

日・米・欧・印のパフォーマンス比較（Cusumano *et. al.*, 2003）

項目	インド	日本	米国	欧州他	合計・平均
調査対象のプロジェクト数（件）	24	27	31	22	104
新規コード行数（KLOC, 中央値）	209	469	270	436	374
出荷後の欠陥密度（中央値）	0.263	0.020	0.400	0.225	0.150

能力成熟度モデル（CMM）のレベルと欠陥密度（Jones 2000）（Davis 2003）

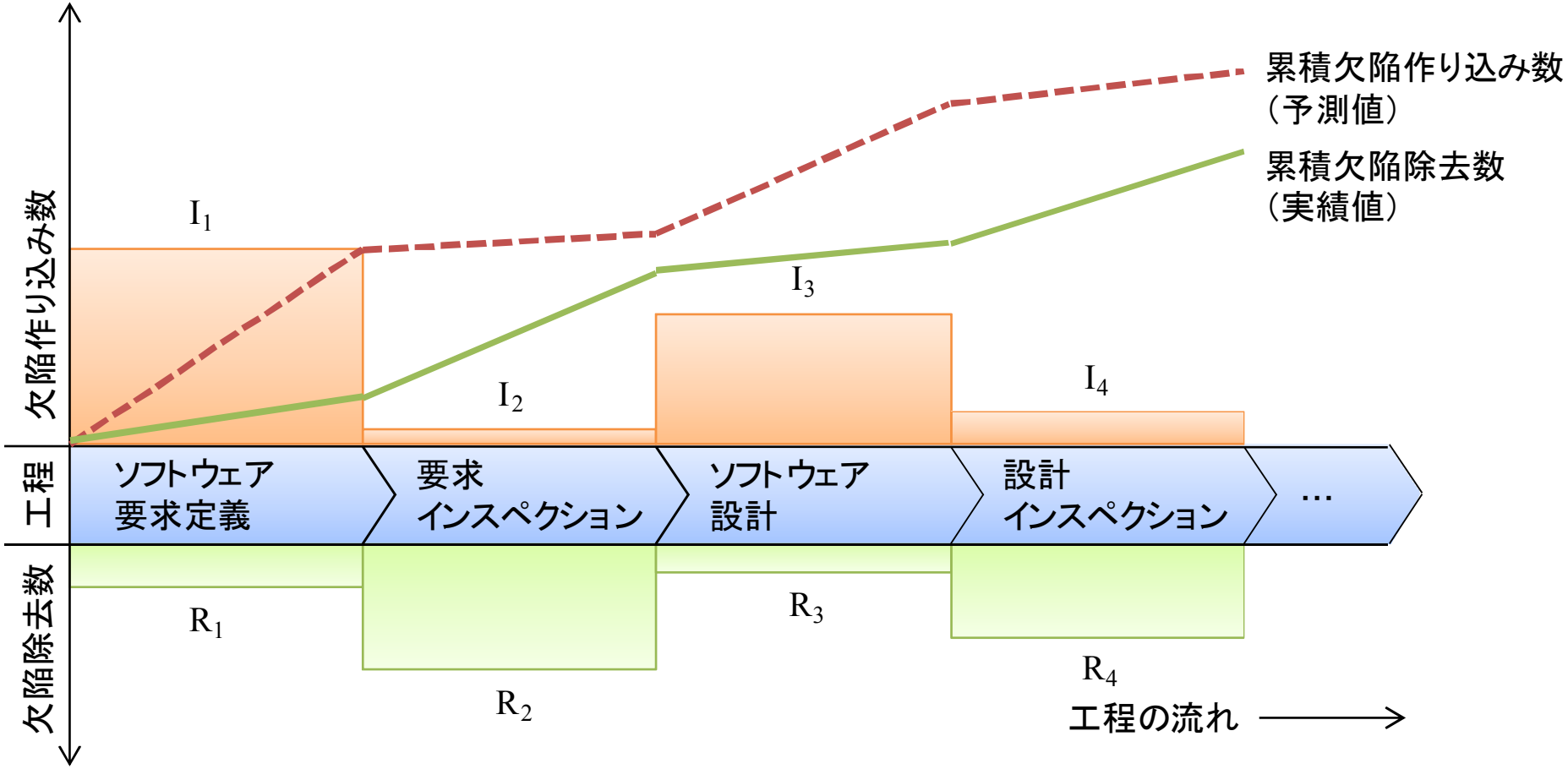
SEI能力成熟度レベル	レベル1	レベル2	レベル3	レベル4	レベル5	TSP
出荷後の欠陥密度	7.50	6.24	4.73	2.28	1.05	0.06

※ TSP ＝ Team Software Process

0.020欠陥/KLOC ＝ 5万行に1件　・・・　満足してよいレベルと言えるか？

Cusumano, M., *et. al.*, "Software Development Worldwide: The State of the Practice," *IEEE Software*, Nov/Dec 2003, pp.28-34.
Jones, C., "*Software Assessments, Benchmarks, and Best Practices*," Addison-Wesley, 2000.
Davis, N. and Mullaney, J., "TSP in Practice: A Summary of Recent Results," *CMU/SEI-2003-TR-14*, 2003.

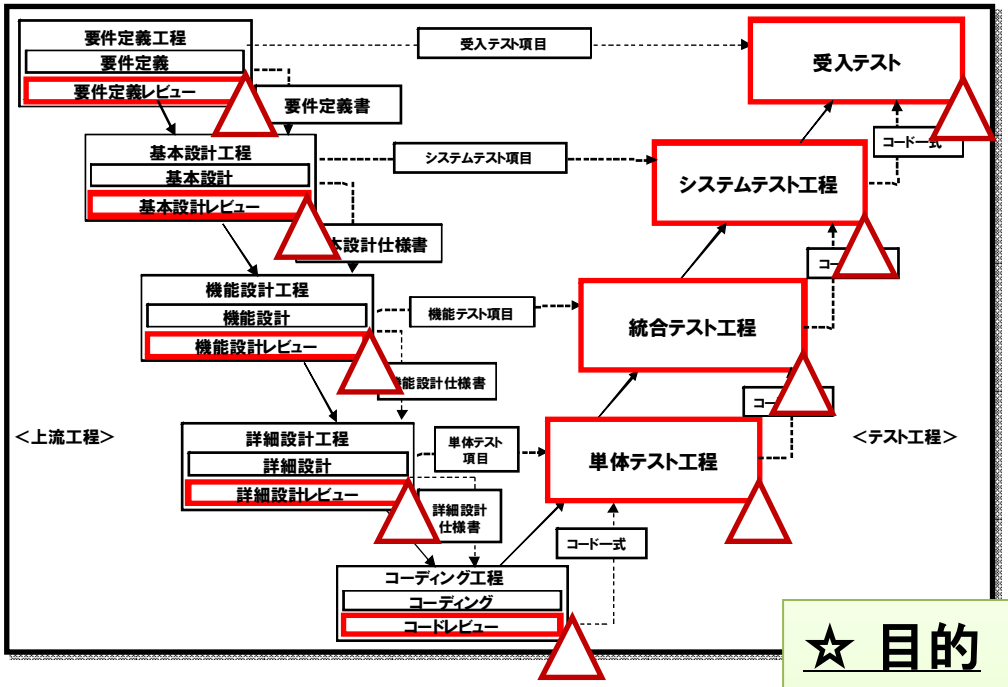
欠陥作り込み・除去モデル



ソフトウェア開発プロセスは、「欠陥を作り込む／除去する」が連続する活動としてモデル化できる

欠陥除去率（DRE）

ソフトウェア開発プロセス



測定項目

・品質
　－ 各レビュー工程での欠陥摘出数
　－ 各工程での欠陥混入数

欠陥・課題記録票

No	該当箇所	指摘内容	…	混入工程
1				
2				

この項目で把握

工程の欠陥除去率

$$\frac{\text{当該工程での欠陥摘出数}}{\text{当該工程での欠陥摘出数} + \text{当該工程での欠陥見逃し数}}$$

☆ 目的

・各欠陥摘出工程の欠陥摘出能力を把握する
・欠陥摘出率の低い工程を重点的に改善する

☆ POINT

・プロジェクト終了後でなければ、測定値が定まらない
・各欠陥の混入工程を記録しておく必要がある
・一般にはDRE(Defect Removal Efficiency)と呼ばれる
　（ソフトウェアメトリクスの分野では基本的なメトリクス）

演習2-1:欠陥除去率を求める

欠陥摘出工程	欠陥作り込み工程			合計
	要求定義	設計	コーディング	
要求レビュー	13			13
設計レビュー	2	12		14
テスト	3	5	32	40
顧客からの報告	1	3	4	8
合計	19	20	36	75

要求レビューの欠陥除去率 = □ / □ = □

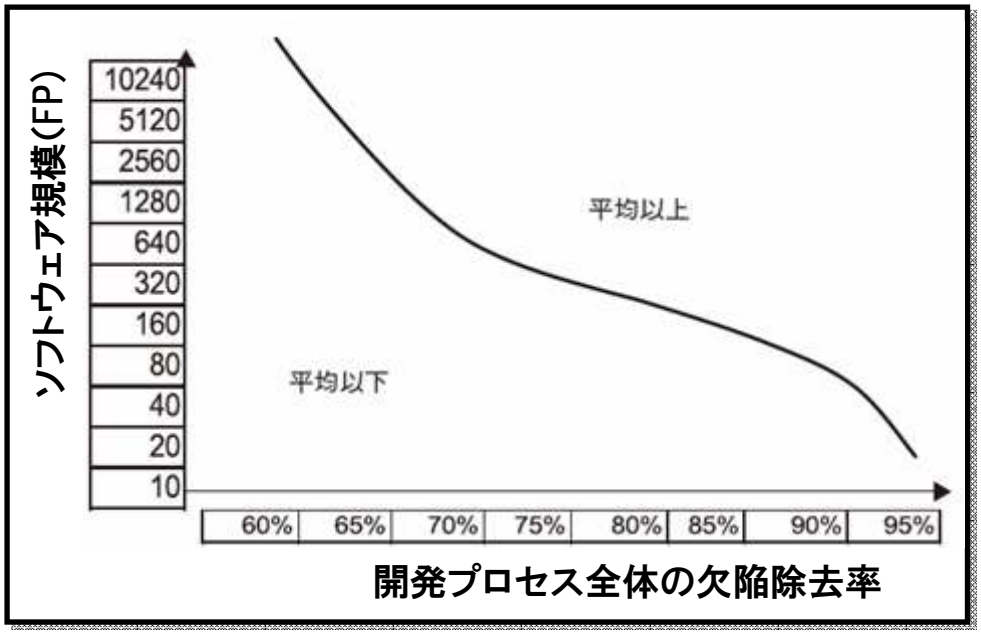
設計レビューの欠陥除去率 = □ / (□ + □ - □) = □

テストの欠陥除去率 = □ / (□ - □ - □) = □

開発プロセス全体の欠陥除去率 = (□ + □ + □) / □ = □

欠陥除去率のベンチマークデータ

Jonesのデータ(1991)



Jonesのデータ(2000)

開発プロセス全体の欠陥除去率
- クラス最高レベル　95%以上
- 米国の平均レベル　80%
- 不適切なプラクティス実施　60%以下

TSP (Team SW Process)の推奨値(2000)

要求インスペクション	70%
設計インスペクション	70%
コードインスペクション	70%

みなさんの組織では、
- 開発プロセス全体の欠陥除去率
- レビュー／インスペクションの欠陥除去率
- テストの欠陥除去率

は、どの程度ですか？

Jones, C., "*Applied Software Measurements*," McGraw-Hill, 1991(鶴保・富野監訳『ソフトウェア開発の定量化手法』構造計画研究所、1993)
Jones, C., "*Software Assessments, Benchmarks, and Best Practices*," Addison-Wesley, 2000.
Humphrey, W. S., The Team Software Process℠ (TSP℠), CMU/SEI-2000-TR- 023, 2000.

演習2-2：欠陥除去率の目標値から、欠陥除去目標を考える

欠陥除去工程	欠陥作り込み工程			合計
	要求定義	設計	コーディング	
要求レビュー	13			13
設計レビュー	2	12		14
テスト	3	5	32	40
顧客からの報告	1	3	4	8
合計	19	20	36	75

欠陥除去率の
目標値を設定

→ 80%

→ 80%

→ 95%

欠陥除去工程	欠陥作り込み工程			合計
	要求定義	設計	コーディング	
要求レビュー				
設計レビュー				
テスト				
顧客からの報告				
合計	19	20	36	75

欠陥除去率の目標値を上の通りに設定場合、各欠陥除去工程で除去すべき欠陥数はいくつになりますか？

ただし、
- 作り込みは変化なし
- 小数点以下は四捨五入
- 見逃し欠陥数は、減少数に比例して後工程に配分する

演習2-2：欠陥除去率の目標値から、欠陥除去目標を考える

欠陥除去工程	欠陥作り込み工程			合計
	要求定義	設計	コーディング	
要求レビュー				
設計レビュー				
テスト				
顧客からの報告				
合計	19	20	36	75

要求レビューの欠陥除去目標 = □ * □ ≒ □

要求定義で作り込んだ欠陥の残り□ 個を、以降の欠陥除去工程に配分

設計レビューの欠陥除去目標 = (□ + □ - □) * □ ≒ □

要求定義で作り込み、設計レビューで除去する□ 個を引いた□ 個が目標

設計で作り込んだ欠陥の残り□ 個を、以降の欠陥除去工程に配分

同様に、テストでの欠陥除去目標を算出する

ここでのまとめ

- 欠陥除去プロセスの能力を表す重要なメトリクス
  - 出荷後の欠陥密度
  - 欠陥除去率(DRE)
- 欠陥密度について
  - 最終成果物の品質を、まずは把握しよう
  - ベンチマークデータと比較し、自組織の立ち位置を確認する
  - 目標値を設定し、達成戦略を考える
- 欠陥除去率について
  - 100%は非現実的
  - 欠陥除去プロセス別の欠陥除去能力を評価する
  - 欠陥除去能力の低いプロセスを重点的に改善する
  - 目標値を設定し、各プロセスでの欠陥除去目標を定める

4. 動的モデルによる欠陥予測

- 動的モデルの考え方
- レイリーモデルによる発見欠陥数の予測
- 欠陥から工数、生産性へ

欠陥数の予測手法とモデル

- 静的モデル: 過去の開発実績に基づく予測
  - 「プロセスやチームが似ていれば、各工程や全体のDREも似たものになるはず」
  - 開始前から予測可☺　過去の類似実績必要☹
  - COCOMO II に基づくCOQUALMO

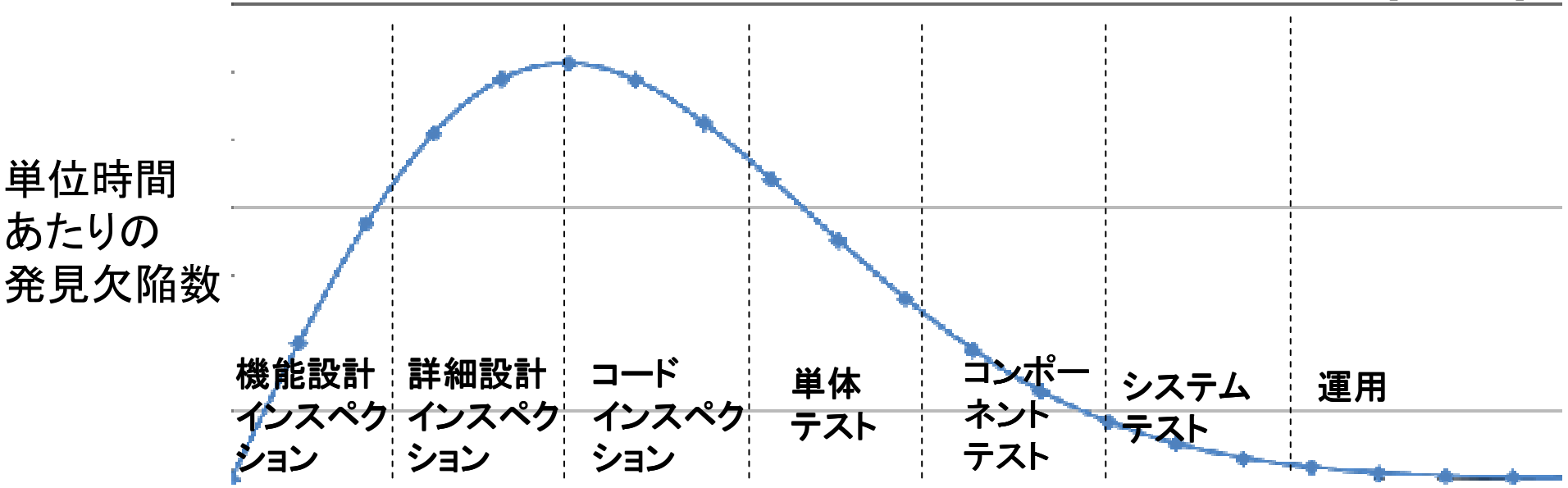
- 動的モデル: 現在の欠陥データに基づく予測
  - 「プロセスやチームの違いに関わらず、欠陥発見数は既知の分布に従うはず」
  - 記録以降で予測可☹　過去の類似実績不要☺
  - ライフサイクル全体: レイリー曲線
  - テスト以降: 指数曲線、S字曲線

Center for Software Engineering, University of Southern California, "COQUALMO," http://csse.usc.edu/csse/research/COQUALMO/

動的モデルの考え方

- 欠陥の見つかり方には、法則性がある
  - 設計までは、時間とともにより多く見つかる
  - 実装以降は、時間とともにより見つかりにくくなる
  - 時間とともにゼロに近づくが、ゼロにはならない
- プロジェクト(さらにはエンジニアリング)に共通

IBM AS 400の場合 [Kan04]



[Kan04] S. Kan 著, 古山・富野 監訳, “ソフトウェア品質工学の尺度とモデル”, 共立出版, 2004

レイリー(Rayleigh)モデル

- **確率密度関数**
  - 時刻 t における単位時間あたりの発見欠陥数

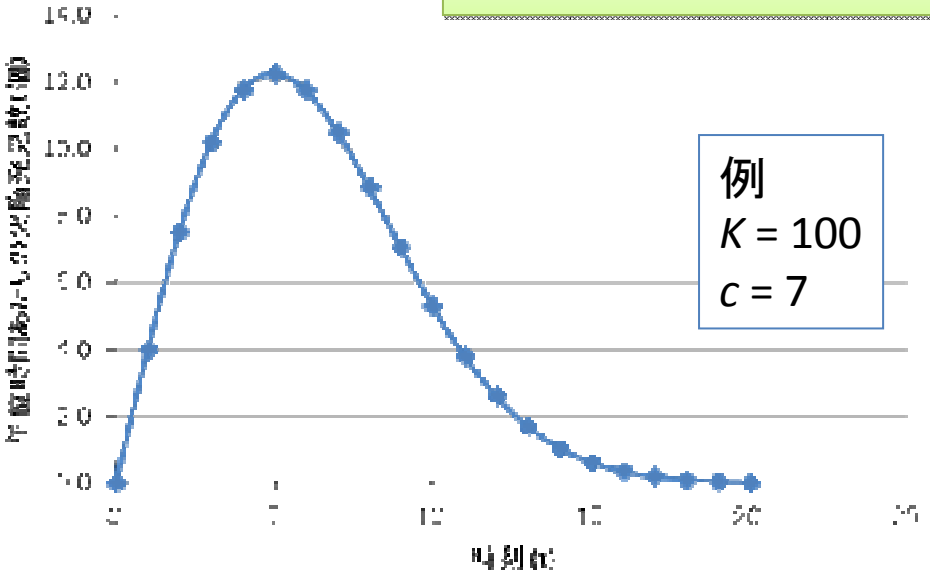
$$f(t) = K \cdot 2\left(t/c^2\right) \cdot e^{-(t/c)^2}$$

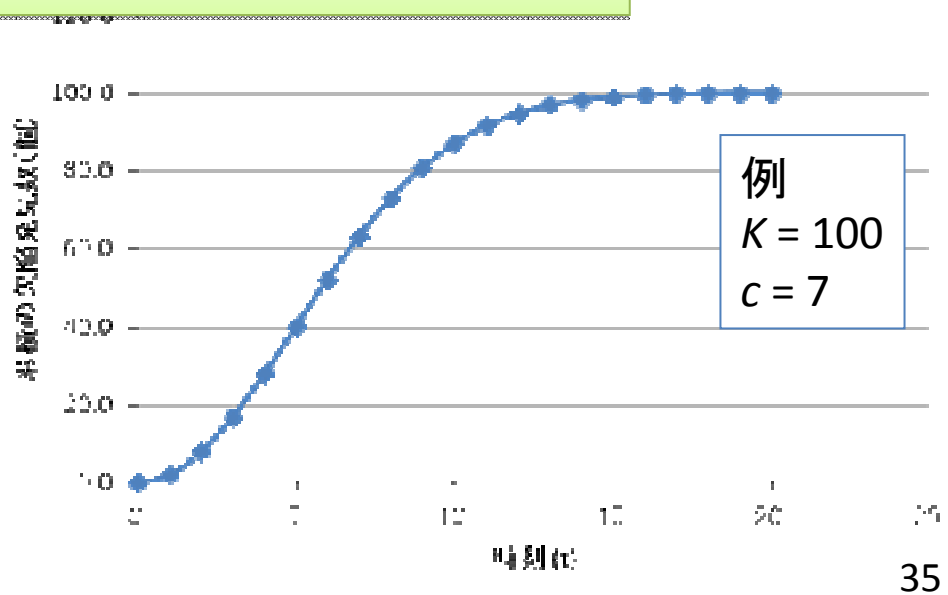
- **累積分布関数**
  - 時刻 t までに発見された欠陥の累積数、f の積分

$$F(t) = K\left(1 - e^{-(t/c)^2}\right)$$

c: 定数　　K: 欠陥の総数

c と K をどのように求めればよいのか？





パラメータを推定しやすいように変形

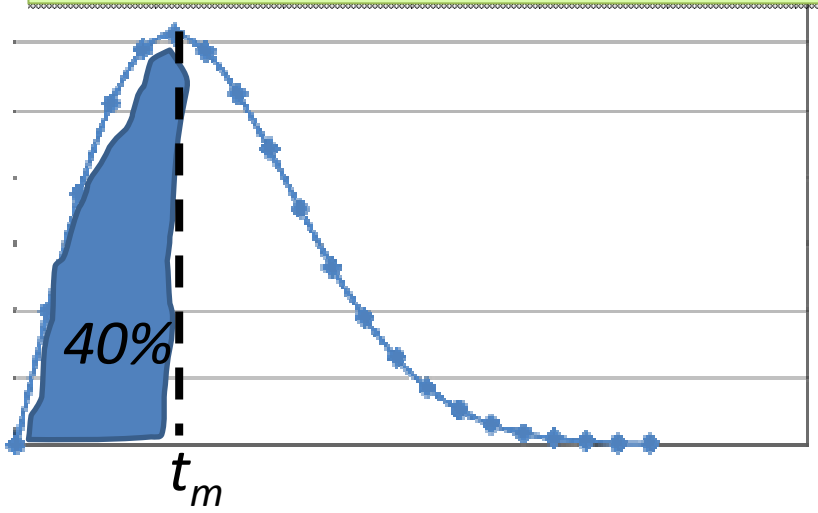
$$f(t) = K \cdot 2\left(t/c^2\right) \cdot e^{-(t/c)^2} \qquad F(t) = K\left(1 - e^{-(t/c)^2}\right)$$

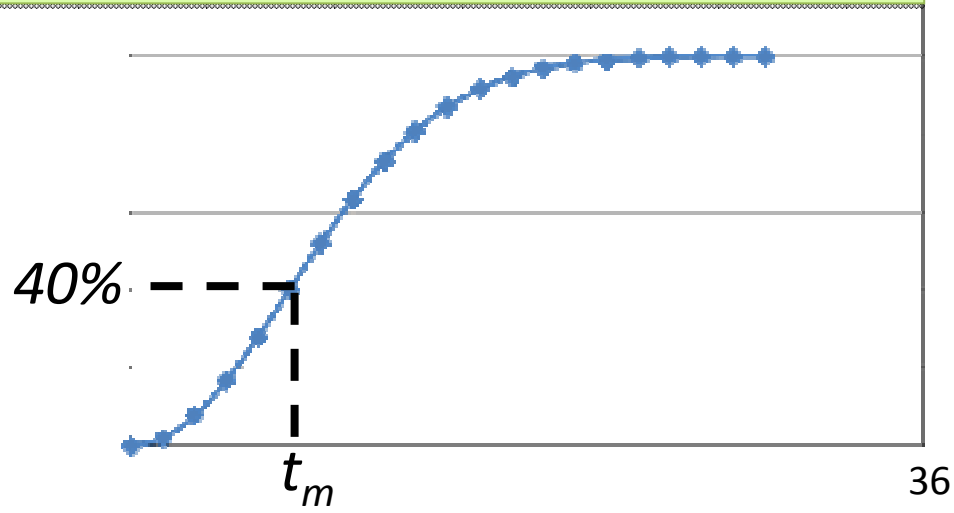
$t_m = f(t)$ が最大となる t とするとき $c = \sqrt{2}t_m$

$$f(t) = K \cdot \left(1/t_m\right)^2 \cdot te^{-\left(1/2t_m^2\right)t^2} \qquad F(t) = K\left(1 - e^{-\left(1/2t_m^2\right)t^2}\right)$$

$$f'(t_m) = 0 \qquad F(t_m) \cong 0.4K$$

単位時間あたり欠陥発見数がピークの t_m さえ分かればよい。
t_m までに累積して総欠陥数の40%が見つかっているはず。





演習: t_mと 総欠陥数40%による予測

- 次のとき、第10週に見つかる欠陥数はいくつか？

週	1	2	3	4	5	6	7	8	9	10
発見欠陥数	20	41	48	52	62	59	52	44	33	？

$$f(t) = K \cdot \left(1/t_m\right)^2 \cdot te^{-\left(1/2t_m^2\right)t^2}$$

- t_m =
- K =
- $f(t) =$
- $f(10) \cong$

手軽だが大雑把。より丁寧には、データポイントをみたい。

演習: t_mと１週目のデータポイントによる予測

- 次のとき、第10週に見つかる欠陥数はいくつか？

週	1	2	3	4	5	6	7	8	9	10
発見欠陥数	20	41	48	52	62	59	52	44	33	？

$$f(t) = K \cdot \left(1/t_m\right)^2 \cdot te^{-\left(1/2t_m^2\right)t^2}$$

- t_m =
- $f(1)$ = $=$
- K =
- $f(10) =$

用いる1データポイントに依存。全データポイントを考慮したい。

演習: t_mと 全データポイントによる予測

- 次のとき、第10週に見つかる欠陥数はいくつか？

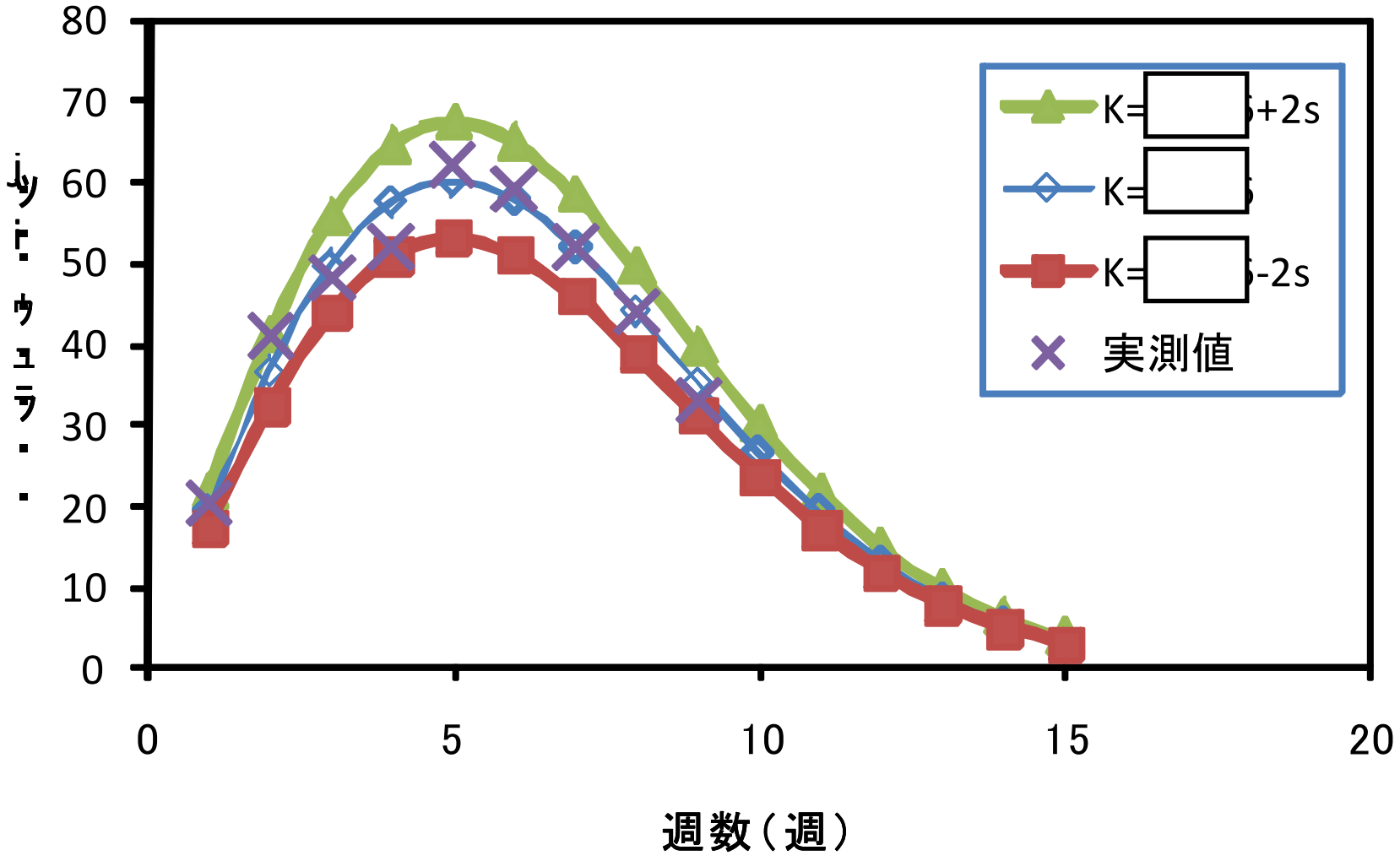
週	1	2	3	4	5	6	7	8	9	10
発見欠陥数	20	41	48	52	62	59	52	44	33	?
K_i		446.9	478.9	516.4	511.1	505.0	494.8	494.5	463.2	

$$f(t) = K \cdot \left(1/t_m\right)^2 \cdot te^{-\left(1/2t_m^2\right)t^2}$$

- t_m =
- K_i =
- K_iの平均 = 標準偏差 σ
- $f(10)$ =

95%信頼水準の予測を得た。過去の類似実績があればより精密に。

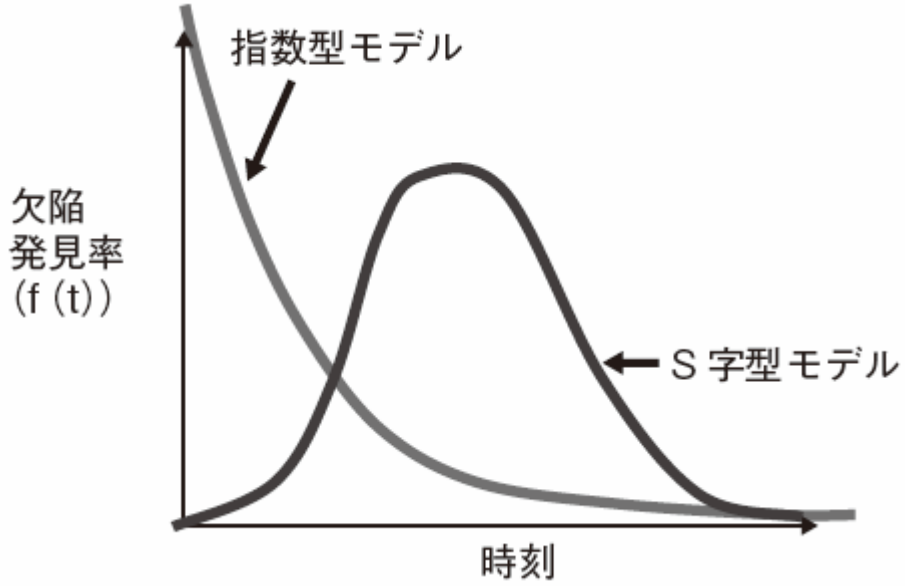
上方・下方管理限界に向けた予測

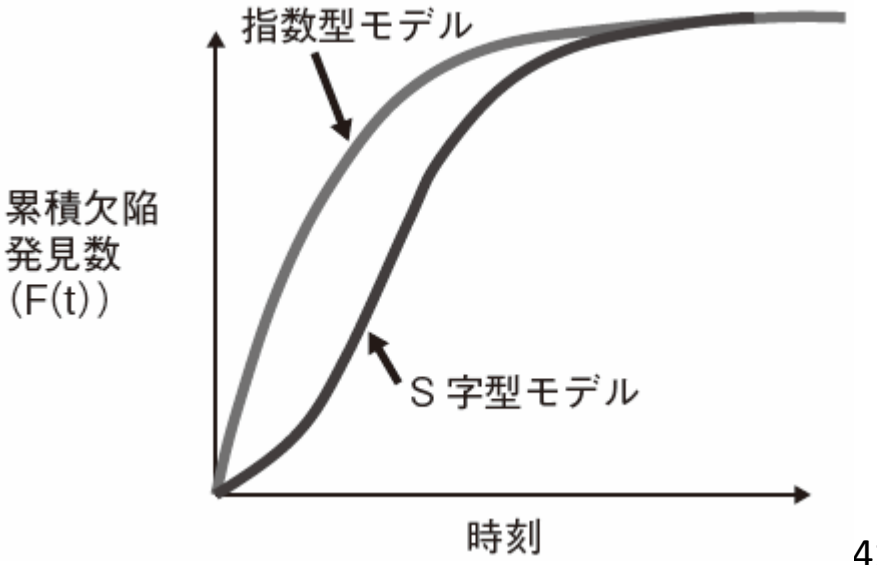


この例では、K＝□±2σ に(ほぼ)すべて収まった

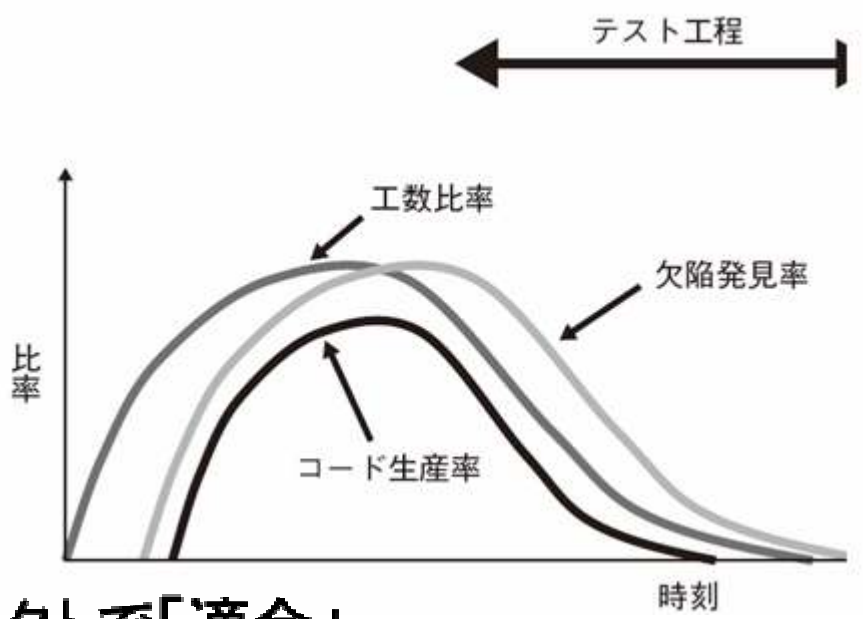
テスト開始以降に特化した予測モデル

- S字型モデル
  - 最初は欠陥発見率がゆっくり上昇
  - その後、ぐっと高くなり、最後にゆるやかに収束
- 指数型モデル
  - もっとも高いところからぐっと減少、その後、緩やかに収束
  - レイリーモデルと同じく、ワイブル(Weibull)曲線の一種

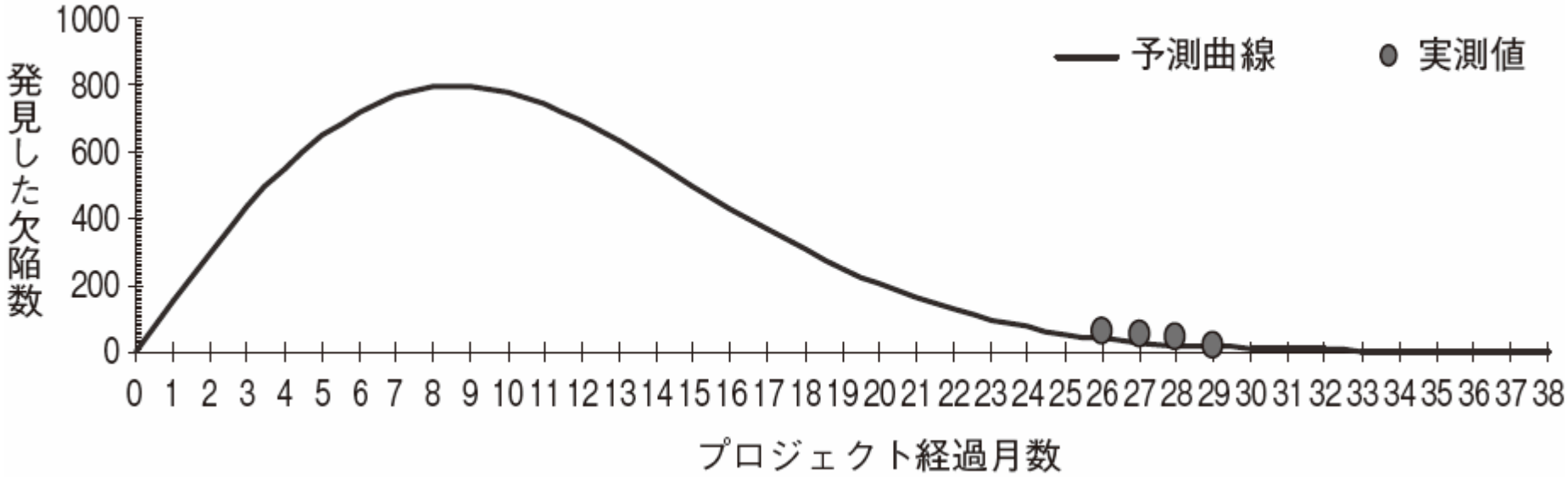




欠陥から工数、生産率へ



- 工数と欠陥数は一般に比例
  - 人々は一定の割合で誤りを犯す
- 動的モデルの利用例
  - タタ・エルクシ社: 20超のプロジェクトで「適合」
  - ノースロップグラップ社: 早期段階のデータからテスト工程の欠陥数を適切に予測



ここでのまとめ

- 欠陥数を予測する2種類のモデル
  - 静的モデル
  - 動的モデル
- 動的モデル
  - 「欠陥の見つかり方には法則性がある」
  - ライフサイクル全体: レイリーモデル
  - テスト以降: S字型モデル、指数型モデル
  - パラメータ推定による予測、特に管理限界を持たせた予測
  - どのような曲線のどの場所にいるのか
- 欠陥から工数へ
  - 工数に続いて欠陥発見率が遅れて、同じカーブ

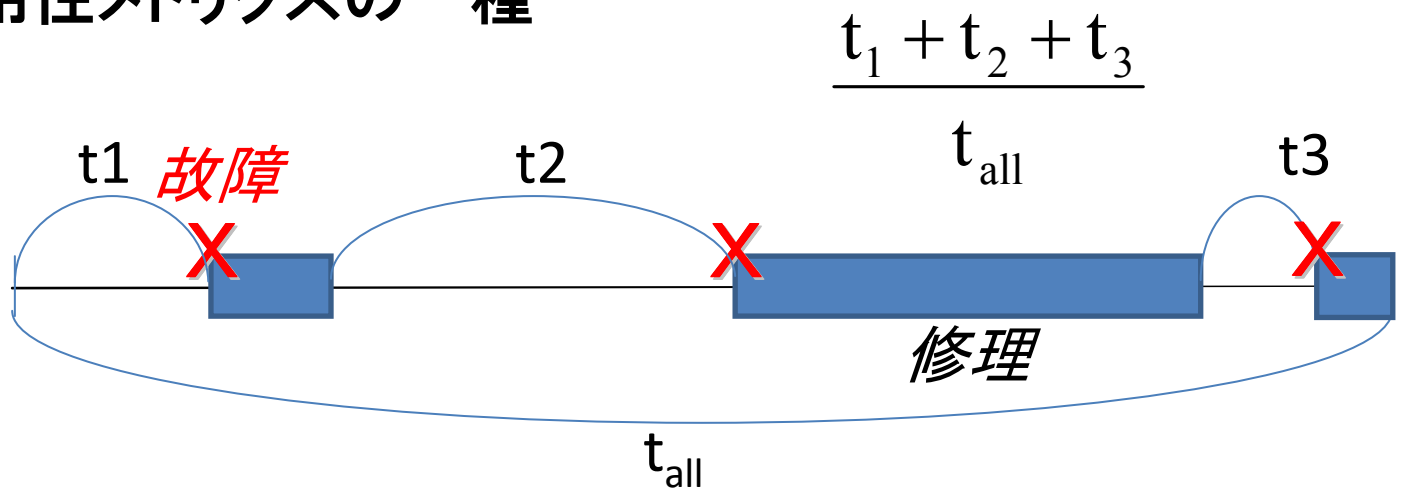
5. システムの可用性や信頼性との関わり

- 可用性と信頼度
- 一様分布とランダム分布
- MTTF, 瞬間故障率の予測: 欠陥数、プロファイルテスト
- 信頼性向上: システム構成、若化

目標としての可用性(Availability)

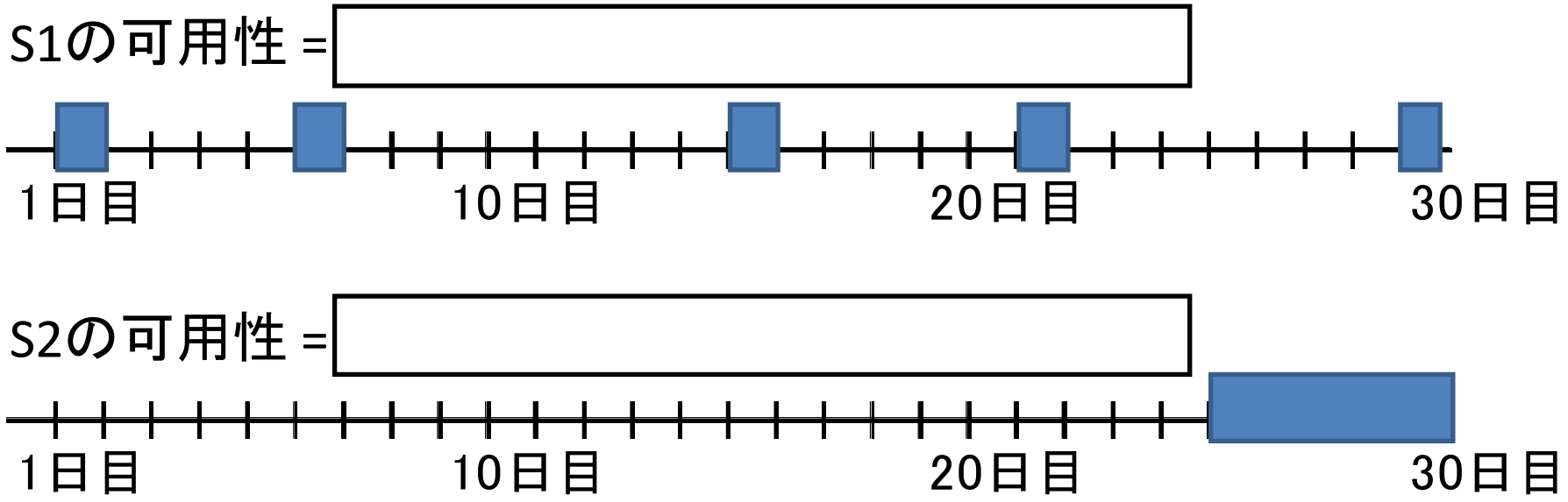
- ユーザの期待は、ゼロ欠陥というよりも、必要なときにシステムが機能するということ。
  - 例: 欠陥がただ一つのシステムがある。出荷してよいか？
- 可用性: システムが正常に機能する確率
- $稼働率 = \frac{動作時間}{総時間}$

可用性メトリクスの一種



演習: 可用性(稼働率)の測定

- システムS1, S2を30日間運用した。可用性は？
  - S1: 2, 7, 16, 22, 30日目に故障。それぞれ復旧に丸1日。
  - S2: 26日目に故障。復旧に5日間。



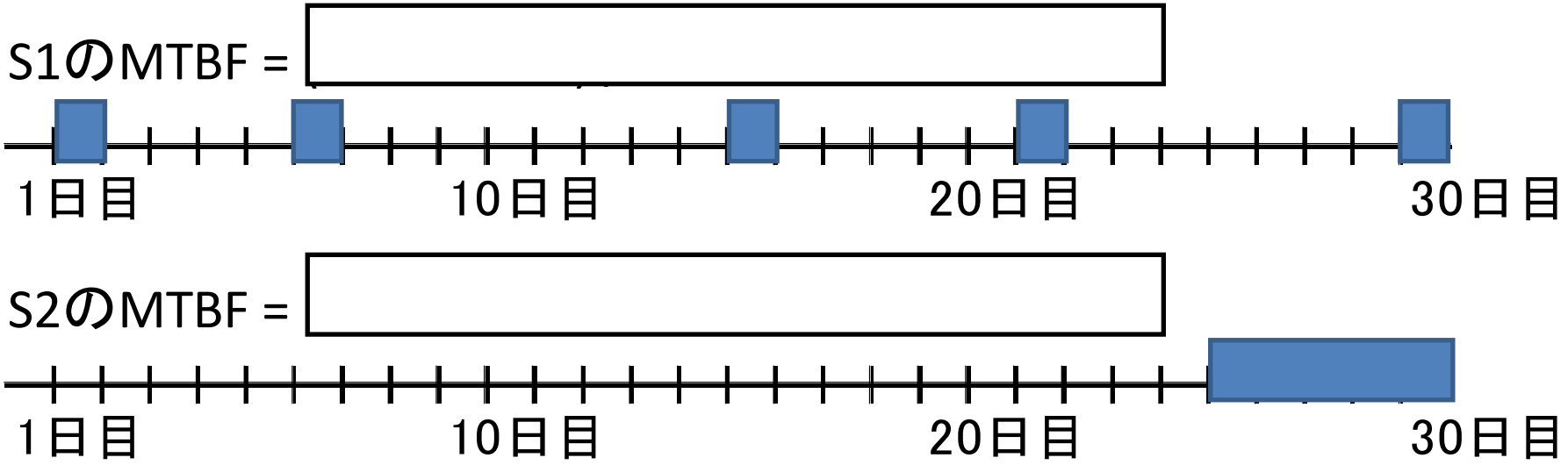
可用性に相違ないが、ユーザへの価値は同等だろうか？

演習: MTBFの測定

- 平均故障間隔 MTBF: Mean Time Between Failures
  - 修理可能システムの故障までの時間平均。信頼性メトリクス。
  - 可用性(稼働率) $= \dfrac{\mathrm{MTBF}}{\mathrm{MTBF} + \mathrm{MTTR}}$

MTTR: Mean Time To Repair
（平均修理時間）

- システムS1, S2を30日間運用した。MTBFは？



S1とS2で信頼性は大きく異なる。特定時刻の信頼性は？

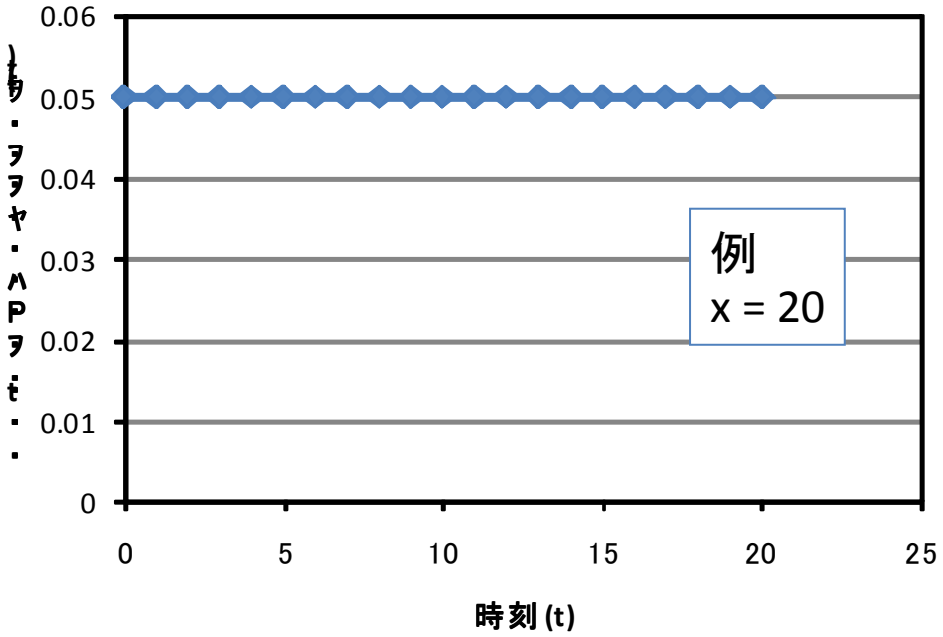
信頼性の予測モデル

- 信頼度: ソフトウェアシステムが所定の環境のもと、特定の期間内に故障することなく機能する確率
- 信頼度成長モデル(Reliability Growth Model)
  - 動的モデル
  - 故障しない確率、欠陥数(特にテスト工程以降)のいずれの予測にも用いる
- 一様分布
  - 必ず特定の時刻xまでに、常に等確率で故障する場合
  - 明快☺　　時刻xが既知とは限らない☹
- ランダム分布(指数分布)
  - 故障が時間経過とともにランダムに発生
  - やや難解☹　何時までに必ず故障するか不明でOK☺

故障モデル: 一様分布

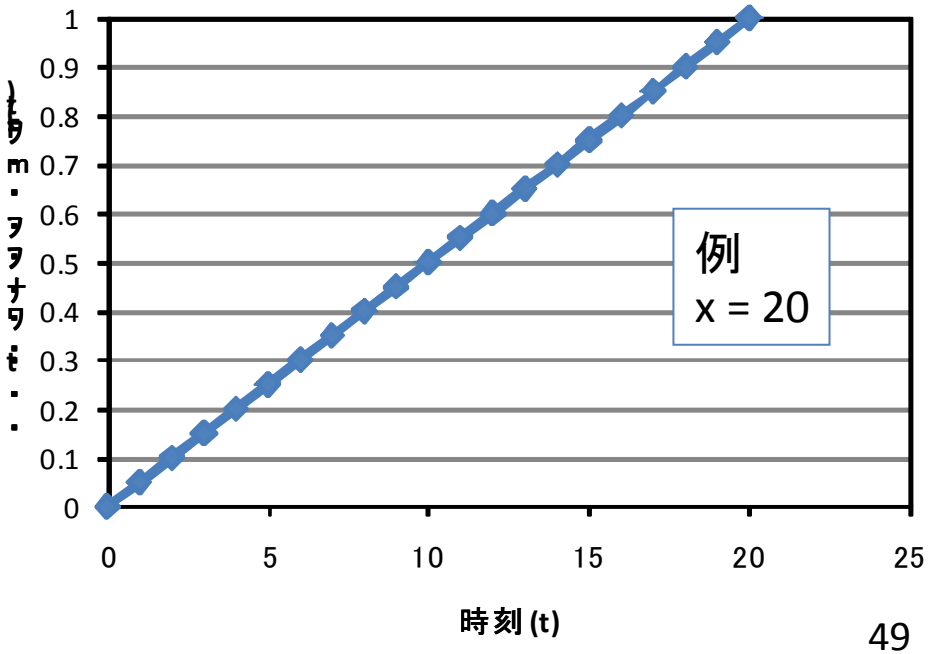
- 必ず時刻 x までに、常に等確率で故障
- 確率密度関数
  - 時刻 t における単位時間あたりの故障率

$$f(t) = 1/x$$



- 累積分布関数
  - 時刻 t までの故障確率
  - f の積分

$$F(t) = t/x \quad (\text{ただし } t \leqq x)$$



故障モデル: 指数分布

- 過去とは独立して故障がランダムに発生し、かつ、時刻 t までに無故障の場合の故障確率が一定(λ)

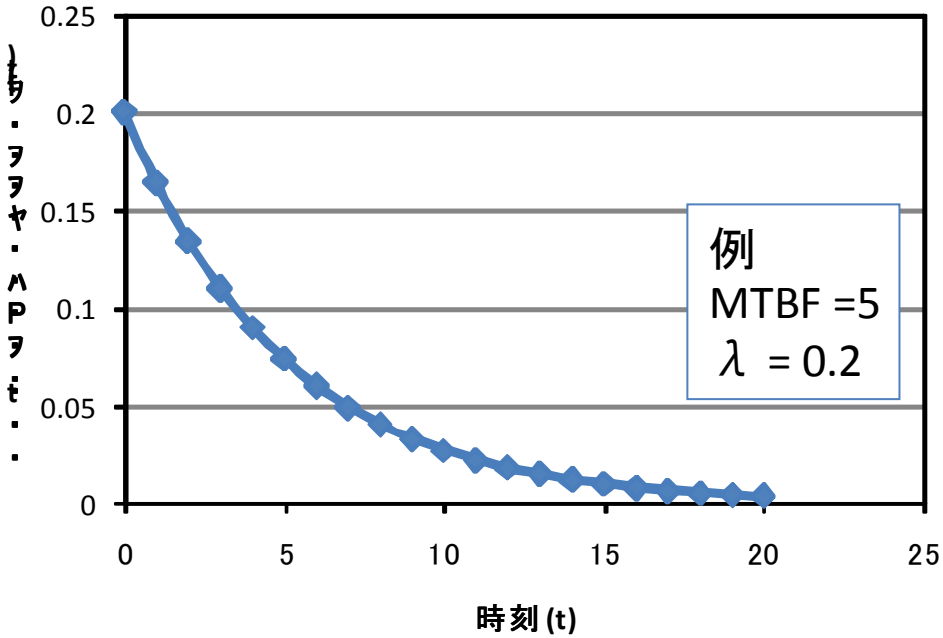
- 確率密度関数

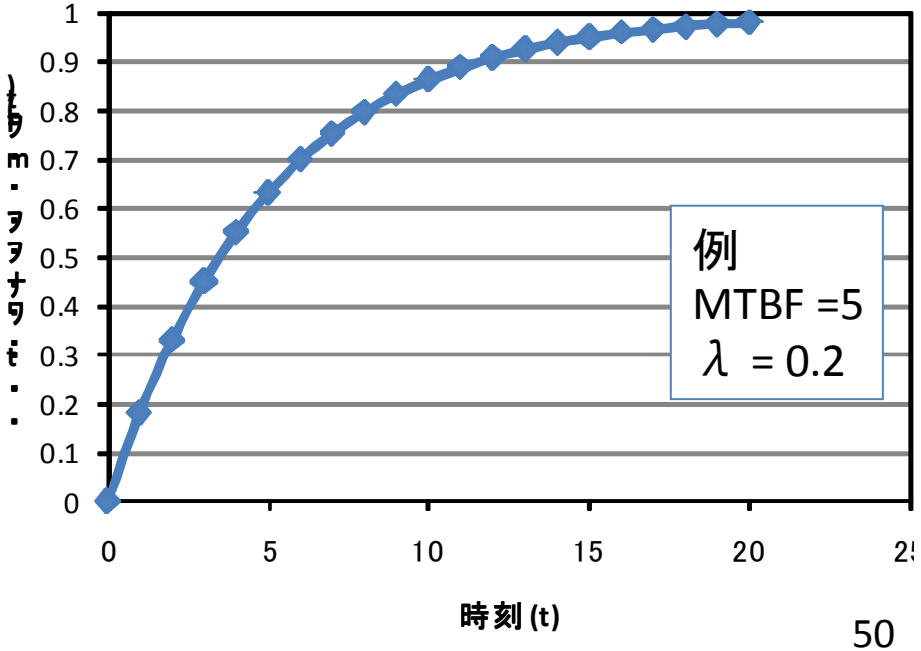
$$f(t) = \lambda e^{-\lambda t}$$

瞬間故障率 $\lambda = \dfrac{1}{\mathrm{MTBF}}$

- 累積分布関数

$$F(t) = 1 - e^{-\lambda t}$$



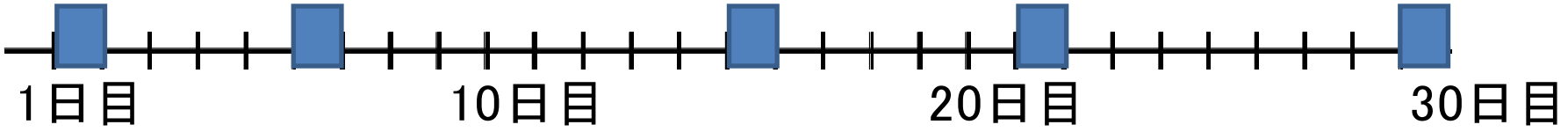


演習: 故障確率と信頼度

- 信頼度: $R(t) = 1 - F(t)$
- システムS1, S2を30日間運用した。故障発生についてランダム分布の場合、10日目終了時点の信頼度は？

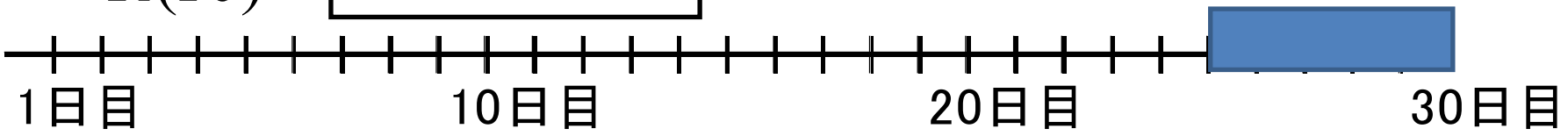
S1: MTBF = □

$R(10) =$ □



S2: MTBF = □

$R(10) =$ □



S1とS2の信頼性の顕著な違いが明らかに

欠陥数によるMTTF(MTBF)の予測

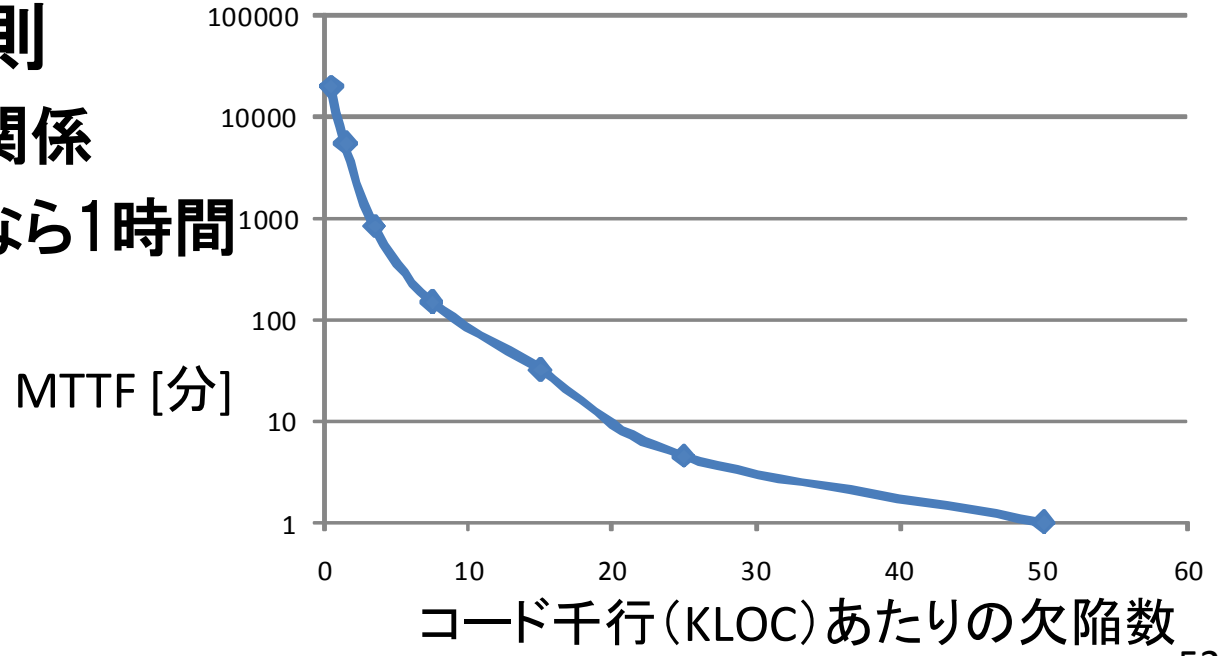
「故障は欠陥箇所の実行により確率的に発生する」

MTTF: Mean Time To Failure(修理不可システムのMTBF)

- ムッサのアルゴリズム: λ = 1/MTTF =

  処理速度[命令数/秒]*実行される欠陥あたり故障数(or 4.2*10-7)*欠陥数/オブジェクトコード命令数

- ジョーンズの経験則
  - 欠陥数とMTTFの関係
  - KLOCあたり10個なら1時間



操作プロファイルテストによるMTTFの予測

- 実際の使用方法と環境に近づけたテスト実行によるリリース後の瞬間故障率 λ = 1/MTTF の予測
  - 操作プロファイル: 使用方法を定量的に特徴づけたデータ

$$\text{初期}\ \lambda = \frac{\sum_i \text{生起確率}(i) * \text{テスト時故障率}(i) * \text{テスト時持続時間}(i)}{\sum_i \text{生起確率}(i) * \text{テスト時持続時間}(i)}$$

ATMの例: (0.007+0.018) / (7+6) ≒ 0.002

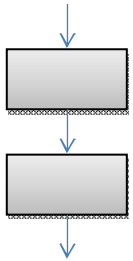
ユースケース	生起確率	平均故障率	平均持続時間
現金を引き出す	0.7	0.001	10
預貯金する	0.3	0.003	20
合計	1		30

演習: システム構成による信頼性向上

- 全体の信頼性はサブシステム群からの構成に依存

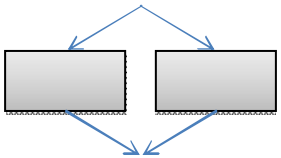
  サブシステムiの信頼度 : $R_i(t)$

- 直列
  - 全て正常動作する確率



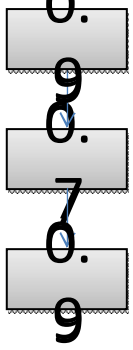
$$R(t) = \prod_i R_i(t)$$

- 並列
  - 1ー（全て故障する確率）

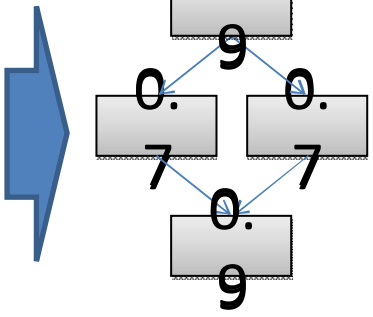


$$R(t) = 1 - \prod_i (1 - R_i(t))$$

- 信頼性を上げるため中段を冗長構成化。信頼度は？



$$0.9 * 0.7 * 0.9$$
$$\cong 0.57$$



その他の向上策

信頼性・可用性の向上

- 耐故障性(フォールトトレラント)設計
  - 例: 縮退運転(デグレード)モード
- 「若化」(Rejuvenation)
  - 再起動やクリーンアップにより経年劣化の防止
  - 例: IBM, 一重冗長構成下の若化で25～60%改善

可用性の向上

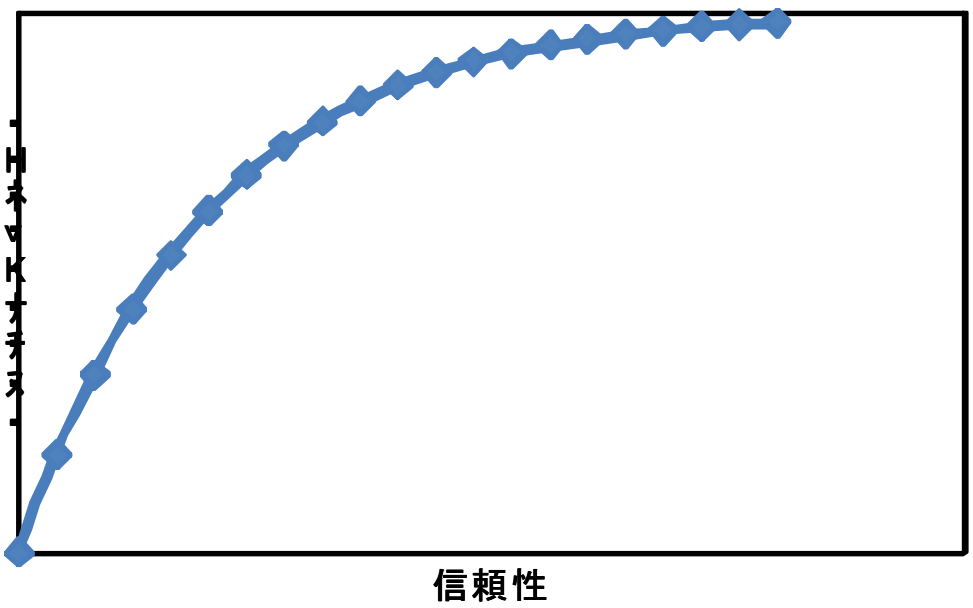
- 平均修理時間(MTTR)短縮
  - 保存・回復の並列化など

$$\text{稼働率} = \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}}$$

信頼性とコスト、リリース時期

- 信頼性はタダではない
  - 例: 信頼性改善に必要な追加工数 E
  - C: 複雑度、k: 係数

$$E(t) = \frac{-kCt}{\log_e R(t)}$$



- いつリリースすべきか？
  - 欠陥数や故障確率が現在、どのような曲線のどのあたりにいるのかを把握したうえで:
  - リリース基準を満たすとき
  - (出荷しないコスト) > (出荷のコスト) が成り立つとき

ここでのまとめ

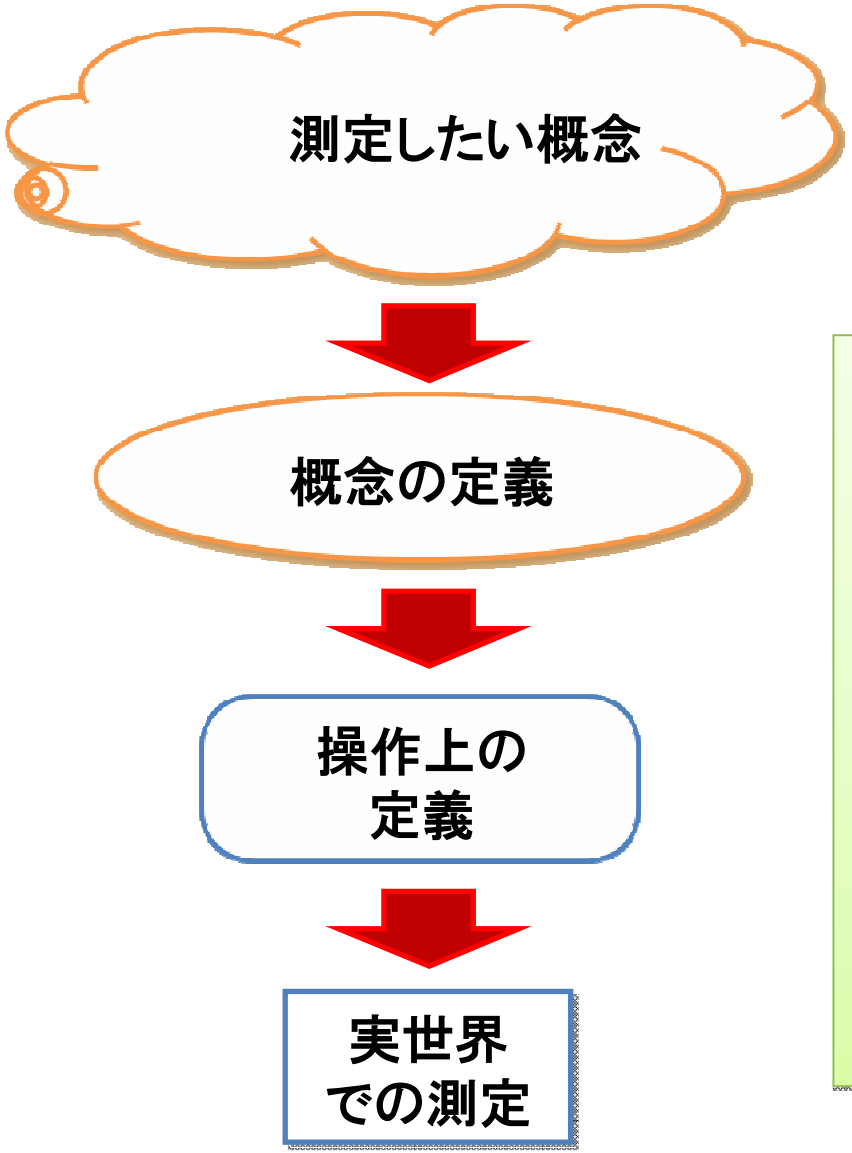
- 可用性: 稼働率
- 信頼性: MTBF(MTTF), MTTR, 故障確率と信頼度
  - 故障モデルの分布形状: 一様分布、ランダム分布
  - 現在、どのような曲線のどのあたりにいるのか
- MTBF(MTTF)・瞬間故障率の予測
  - 欠陥数による予測: ムッサのアルゴリズム、経験則
  - 操作プロファイルテストによる予測
- 信頼性向上に向けて
  - システム構成: 直列、並列
  - 耐故障性設計、若化
- 信頼性とコスト
  - 信頼性はタダではない。対数関係。
  - リリース基準

6. まとめ

取り上げた事柄

- メトリクスの基本概念と必要性: 定量的管理、欠陥
- プロジェクト進捗の把握: テストケース数、テスト管理図、欠陥バックログ、計画
- プロセスの欠陥除去能力: 欠陥密度、欠陥除去率
- 動的モデルによる欠陥予測: レイリーモデル、欠陥から工数へ
- システムの可用性と信頼性: 稼働率、信頼度、MTBF, MTTF, 操作プロファイルテスト

もう一度: メトリクスの限界を知り、意思決定に役立てる

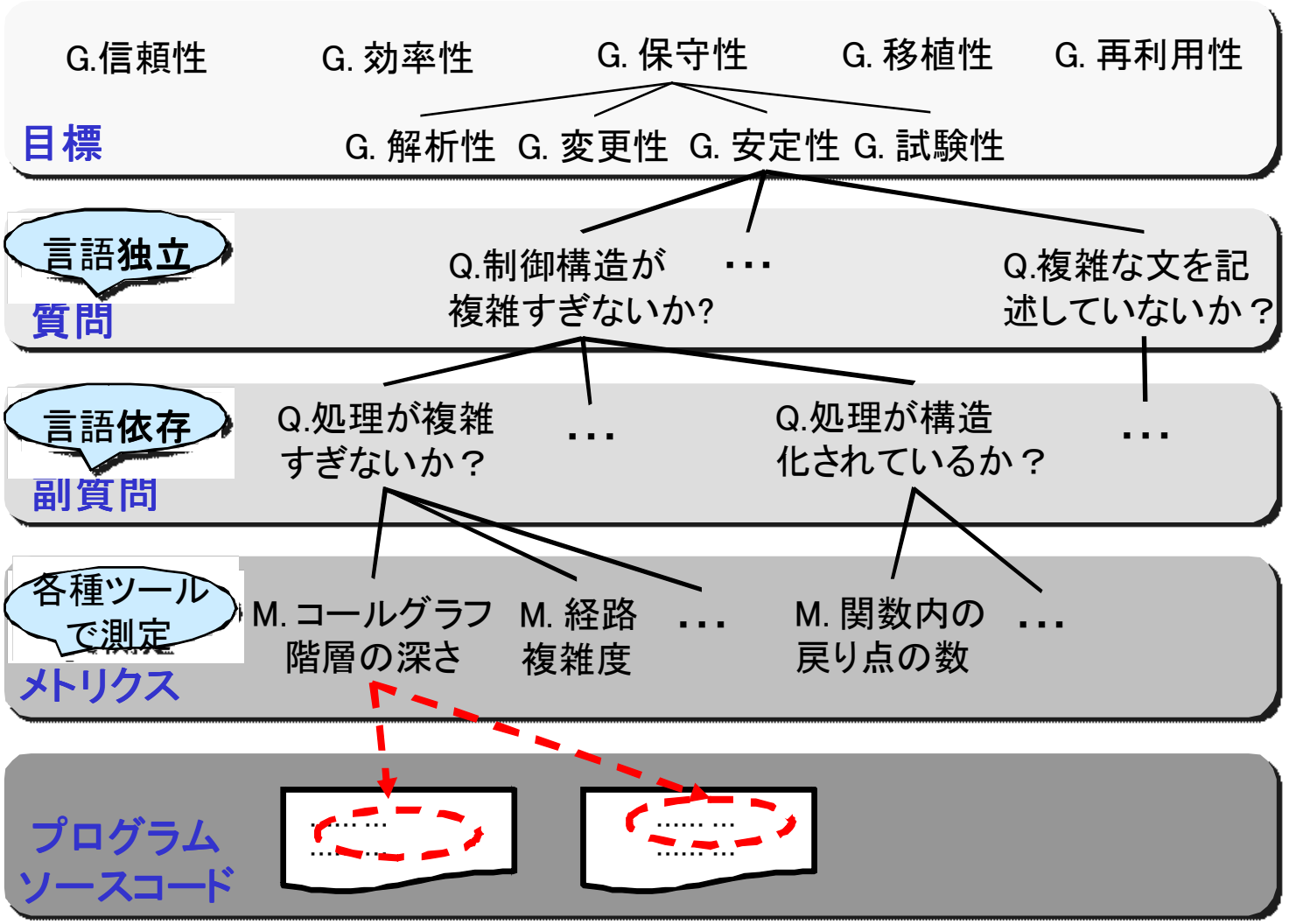


- 「実世界で測定」できることは、「測定したい概念」のごく一部に過ぎない
- 過度の期待を抱いてはいけない
- メトリクスの限界を理解した上で、“informed decision”に役立てる

L. M. Laird and M. C. Brennan, *Software Measurement and Estimation: A Practical Approach*, John-Wiley and Sons, 2006.
(野中・鷲崎訳:演習で学ぶソフトウェアメトリクスの基礎、日経BP社(2009))

事例: 概念から実世界までの対応付け

■ Goal-Question-Metric(GQM)法による段階的マッピング



鷲崎弘宜, 波木理恵子, 福岡呂之, 原田洋子, 渡辺博之, "プログラムソースコードのための実用的な品質評価枠組み", 情報処理学会論文誌, Vol.48, No.8, 2007.

鷲崎弘宜,田邉浩之, 小池利和,"ソースコード解析による品質評価の仕組み", 日経エレクトロニクス,, 招待論文, 2009年1月25日号

おわりに

- ソフトウェアの信頼性向上のために、テスト結果としての欠陥データをグラフで「可視化」し、状況を正しく把握できるようにする。
- 欠陥データを蓄積し、次回以降プロジェクトの予測に活用する。
- プロジェクト内での欠陥データに予測モデルを適用し、予測プロセスの定型化と、複数モデルによるトライアンギュレーションを目指す。
- 経験則により欠陥データを、システム全体の可用性・信頼性の予測へと役立てる。予測には欠陥データと同様にモデルを利用できる。
- 主要なターゲットは「ソフトウェアプロダクト品質」であることを忘れてはいけない。プロダクト品質とプロセス品質を結びつけたうえでのメトリクスの検討が重要。

演習回答

演習1:コード結合計画と欠陥発見計画

週	コード結合 (KLOC)	欠陥修正要員(人)	要員時間配分比率	当該週結合コードの欠陥発見数 4	5	6	7	8	9	10	11	欠陥発見 週別	累積	欠陥修正 週別	累積	欠陥バックログ
1	0	0	0%									0	0	0		
2	0	0	0%									0	0	0		
3	0	0	0%									0	0	0		
4	10	0	0%									0	0	0	0	0
5	20	6	0%	3								3	3	0	0	3
6	30	6	0%	4	6							10	13	0	0	13
7	80	6	20%	2	7	9						18	31	6	6	25
8	20	6	20%	1	4	11	24					40	71	6	12	59
9	10	6	50%	1	2	6	28	6				43	114	15	27	87
10	5	6	50%		1	3	16	7	3			30	144	15	42	102
11	2	6	80%			2	8	4	4	2		20	164	24	66	98
12	0	6	80%				4	2	2	2	1	11	175	24	90	85
13	0							1	1	1	1	4	179	30	120	59
14	0								1	1		2	181	30	150	31
15	0											0	181	30	180	1
16	0	6	100%									0	181	30	210	0
合計	177			11	20	31	80	20	11	6	2					

第4週に結合した10KLOCに含まれる10個の欠陥が、第5週目以降から発見される

第5週結合の20KLOC

欠陥バックログが発生する

第4週に結合した10KLOCには1欠陥/KLOC * 10KLOC =10個の欠陥が含まれると予想できる

演習2-1:欠陥除去率を求める

欠陥摘出工程	欠陥作り込み工程			合計
	要求定義	設計	コーディング	
要求レビュー	13			13
設計レビュー	2	12		14
テスト	3	5	32	40
顧客からの報告	1	3	4	8
合計	19	20	36	75

要求レビューの欠陥除去率 = 13 / 19 = 0.684

設計レビューの欠陥除去率 = 14 / (19 + 20 - 13) = 0.538

テストの欠陥除去率 = 40 / (75 - 13 - 14) = 0.833

開発プロセス全体の欠陥除去率 = (13 + 14 + 40) / 75 = 0.893

演習2-2:欠陥除去率の目標値から、欠陥除去目標を考える

欠陥除去工程	欠陥作り込み工程			合計
	要求定義	設計	コーディング	
要求レビュー	15			15
設計レビュー	1	18		19
テスト	2	1	36	39
顧客からの報告	1	1	0	2
合計	19	20	36	75

要求レビューの欠陥除去目標 = 19 * 0.80 ≒ 15
要求定義で作り込んだ欠陥の残り 4 個を、以降の欠陥除去工程に配分

設計レビューの欠陥除去目標 = (19 + 20 - 15) * 0.80 ≒ 19
要求定義で作り込み、設計レビューで除去する 1 個を引いた 18 個が目標
設計で作り込んだ欠陥の残り 2 個を、以降の欠陥除去工程に配分

同様に、テストでの欠陥除去目標を算出する

演習: t_mと 総欠陥数40%による予測

- 次のとき、第10週に見つかる欠陥数はいくつか？

週	1	2	3	4	5	6	7	8	9	10
発見欠陥数	20	41	48	52	62	59	52	44	33	？

$$f(t) = K \cdot \left(1/t_m\right)^2 \cdot te^{-\left(1/2t_m^2\right)t^2}$$

- t_m = 5 週目
- K = (20+41+48+52+62) / 0.4 = 557.5
- $f(t) = 557.5\left(t/25\right)e^{-t^2/50} = 22.3te^{-t^2/50}$
- $f(10) \cong 30$

手軽だが大雑把。より丁寧には、データポイントをみたい。

演習: t_mと１週目のデータポイントによる予測

- 次のとき、第10週に見つかる欠陥数はいくつか？

週	1	2	3	4	5	6	7	8	9	10
発見欠陥数	20	41	48	52	62	59	52	44	33	？

$$f(t) = K \cdot \left(1/t_m\right)^2 \cdot te^{-\left(1/2t_m^2\right)t^2}$$

- t_m = 5 週目
- $f(1) = K\left(1/25\right)e^{-1/50} = 20$
- $K = 510.1$
- $f(10) = 510.1 \cdot \left(10/25\right) \cdot e^{-10^2/50} \cong 27.6$

用いる1データポイントに依存。全データポイントを考慮したい。

演習: t_mと 全データポイントによる予測

- 次のとき、第10週に見つかる欠陥数はいくつか？

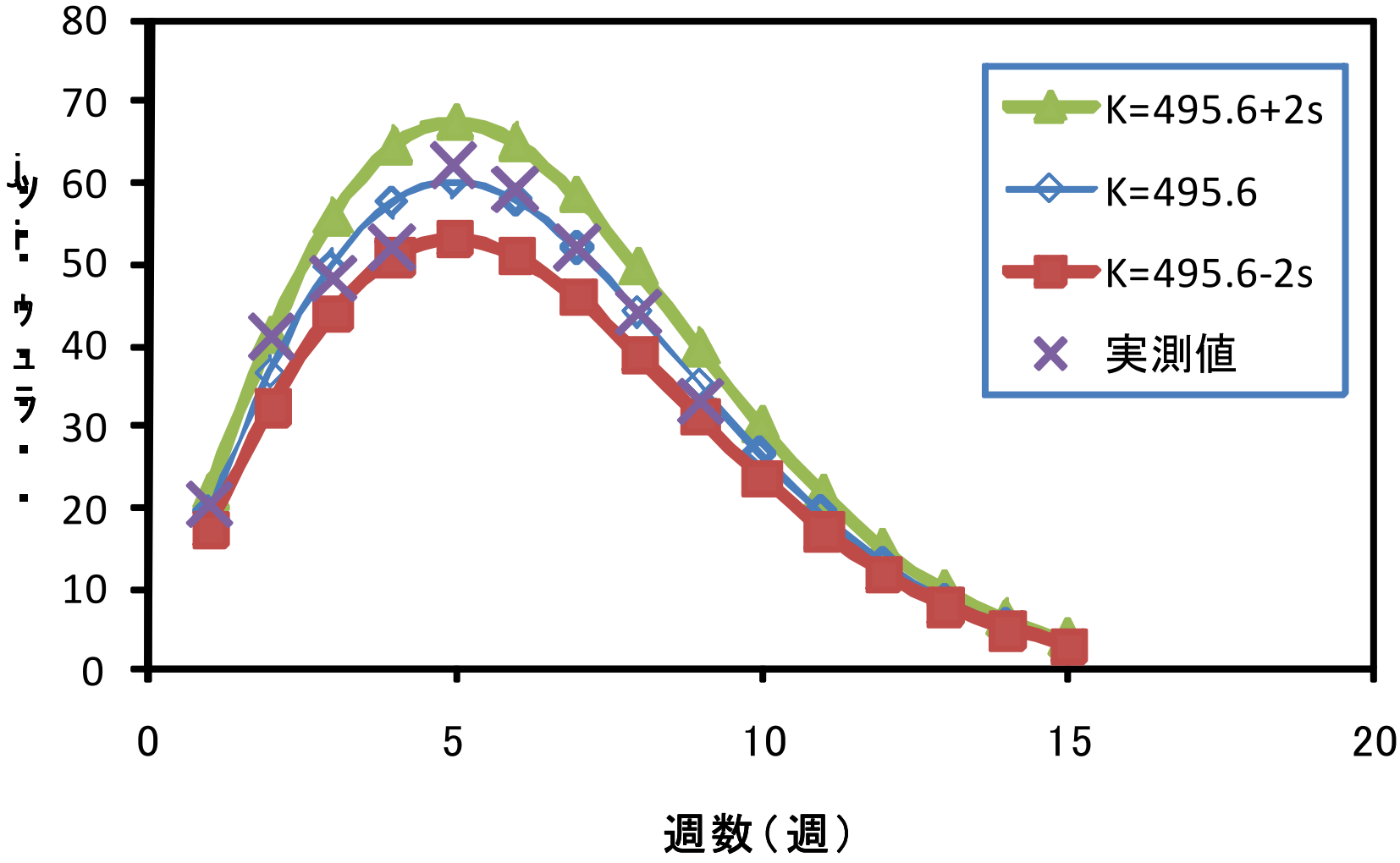
週	1	2	3	4	5	6	7	8	9	10
発見欠陥数	20	41	48	52	62	59	52	44	33	？
K_i	510.1	446.9	478.9	516.4	511.1	505.0	494.8	494.5	463.2	

$$f(t) = K \cdot \left(1/t_m\right)^2 \cdot te^{-\left(1/2t_m^2\right)t^2}$$

- t_m = 5 週目
- $K_i = \dfrac{25}{t} e^{t^2/50} \cdot f(i)$
- K_iの平均 = 495.6　　標準偏差 σ = 29.3
- $f(10) = (495.6 \pm 2\delta) \cdot \left(10/25\right) \cdot e^{-10^2/50} = 23.7 \sim 30.0$

95%信頼水準の予測を得た。過去の類似実績があればより精密に。

上方・下方管理限界に向けた予測

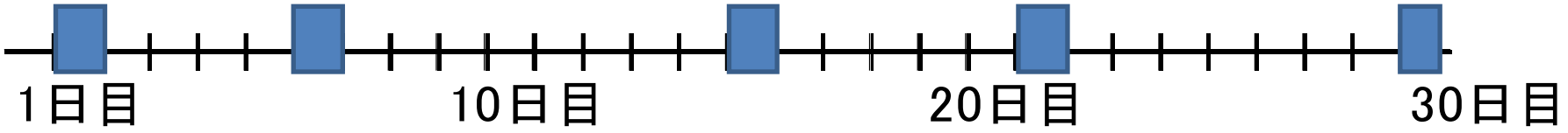


この例では、K =495.6±2σ に(ほぼ)すべて収まった

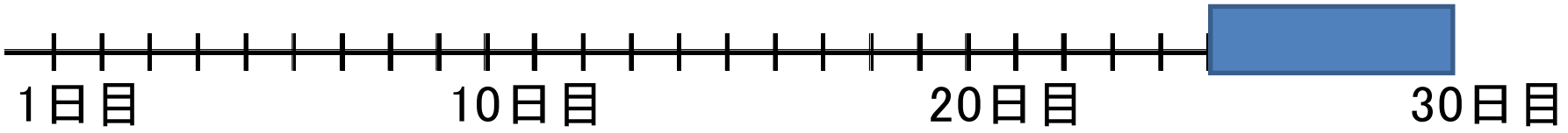
演習: 可用性(稼働率)の測定

- システムS1, S2を30日間運用した。可用性は？
  - S1: 2, 7, 16, 22, 30日目に故障。それぞれ復旧に丸1日。
  - S2: 26日目に故障。復旧に5日間。

S1の可用性 = (1+4+8+5+7) / 30 = (30-5) / 30 = 0.83



S2の可用性 = 25 / 30 = 0.83



可用性に相違ないが、ユーザへの価値は同等だろうか？

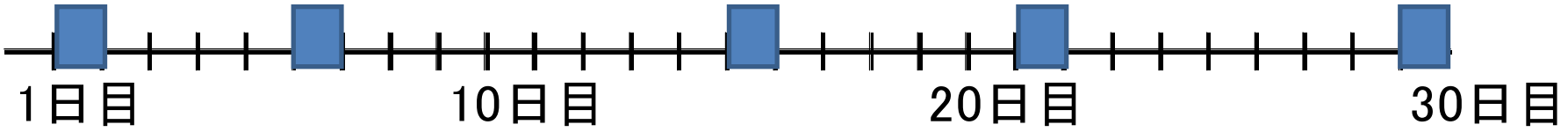
演習: MTBFの測定

- 平均故障間隔 MTBF: Mean Time Between Failures
  - 修理可能システムの故障までの時間平均。信頼性メトリクス。
  - 可用性(稼働率) $= \frac{\mathrm{MTBF}}{\mathrm{MTBF} + \mathrm{MTTR}}$

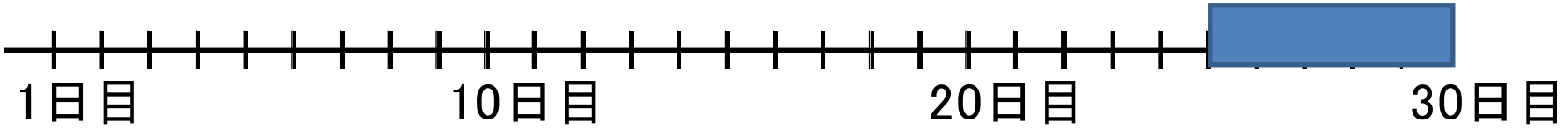
MTTR: Mean Time To Repair
（平均修理時間）

- システムS1, S2を30日間運用した。MTBFは？

S1のMTBF = (1+4+8+5+7) / 5 = 5



S2のMTBF = 25 / 1 = 25



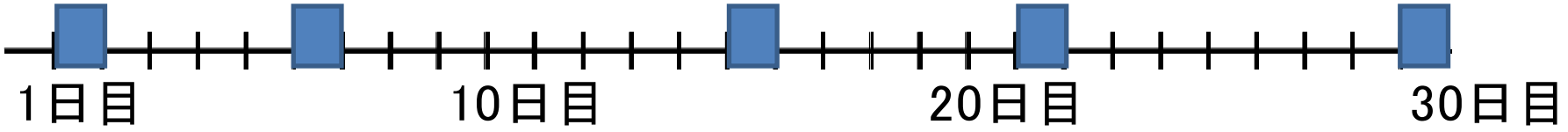
S1とS2で信頼性は大きく異なる。特定時刻の信頼性は？

演習: 故障確率と信頼度

- 信頼度: $R(t) = 1 - F(t)$
- システムS1, S2を30日間運用した。故障発生についてランダム分布の場合、10日目終了時点の信頼度は？

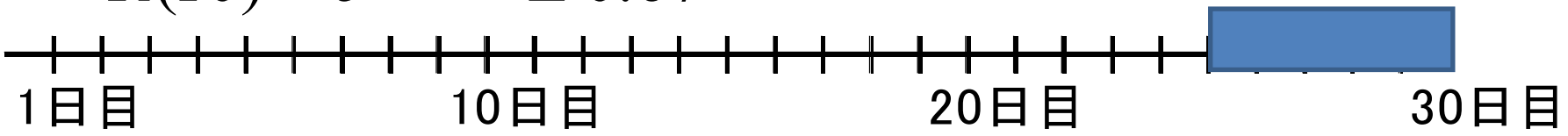
S1: MTBF = 5

$$R(10) = e^{-0.2 \cdot 10} \cong 0.14$$



S2: MTBF = 25

$$R(10) = e^{-0.04 \cdot 10} \cong 0.67$$



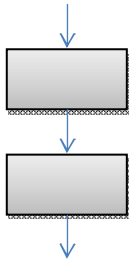
S1とS2の信頼性の顕著な違いが明らかに

演習: システム構成による信頼性向上

- 全体の信頼性はサブシステム群からの構成に依存

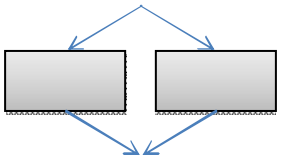
  サブシステムiの信頼度 : $R_i(t)$

- 直列
  - 全て正常動作する確率



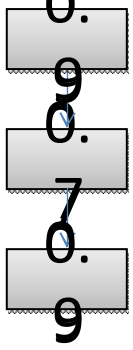
$$R(t) = \prod_i R_i(t)$$

- 並列
  - 1ー（全て故障する確率）

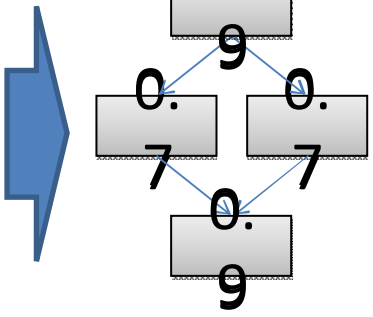


$$R(t) = 1 - \prod_i (1 - R_i(t))$$

- 信頼性を上げるため中段を冗長構成化。信頼度は？



$$0.9 * 0.7 * 0.9$$
$$\cong 0.57$$



$$0.9 * (1 - 0.3 * 0.3) * 0.9$$
$$\cong 0.73$$