

ソフトウェアテスト

[12] 品質管理とメトリクス

Software Testing
[12] Software Quality Management
and Software Metrics

あまん ひろひさ
阿萬 裕久(AMAN Hirohisa)
aman@ehime-u.ac.jp

(C) 2007-2023 Hirohisa AMAN

1

品質管理(ひんしつかんり): quality management

品質の概念

□ 品質とは？

その製品やサービスが要求を満たしている程度

- 製品やサービスには「**満たすべき事項**」や「**期待される事項**」がある: それらの**満足度合い**
- なお, 明記されていないが「できて当たり前」という事項も含まれる

(C) 2007-2023 Hirohisa AMAN

2

品質(ひんしつ): quality

概念(がいねん): concept, notion

製品(せいひん): product

サービス: service

要求を満たしている程度(ようきゅうをみたしているていど): the extent to which it satisfies the requirements

満たすべき事項(みたすべきじこう): the things to be satisfied (met)

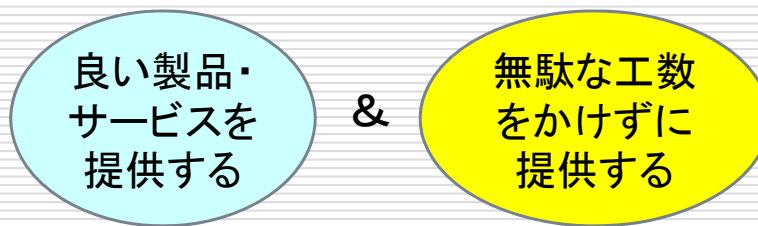
期待される事項(きたいされるじこう): the things to be expected

満足度合い(まんぞくどあい): the extent to which it satisfies

当たり前(あたりまえ): must-be

品質管理

- 顧客の要求を満たすように製品やサービスを提供する活動であり, なおかつ
- それを経済的に提供する活動



(C) 2007-2023 Hirohisa AMAN

3

品質管理(ひんしつかんり): quality control

顧客の要求(こきやくのようきゅう): requirements from the customers (users)

提供する(ていきょうする): supply

活動(かつどう): activity

経済的(けいざいてき): economically

無駄な工数(むだなこうすう): wasted effort

工場での大量生産の例(1／2)

【良い製品・サービスを提供する】

- 大量生産する中で**不良品**が一部含まれてしまうことがある
- 製品の**検査**を行って不良品を検出・除外し、顧客の要求を満たしたもののみを出荷する

適切な検査体制を整えることは重要な品質管理活動

(C) 2007-2023 Hirohisa AMAN

4

工場(こうじょう): factory

大量生産(たいりょうせいさん): mass production

不良品(ふりょうひん): defective product

検査(けんさ): inspection

検出する(けんしゅつする): detection

除外する(じょがいする): exclude

出荷する(しゅっかする): make shipment

適切な(てきせつな)検査体制(けんさたいせい)を整える(ととのえる): develop a proper inspection system

重要な(じゅうような): important

品質管理活動(ひんしつかんりかつどう): quality control activity

工場での大量生産の例(2/2)

【無駄な工数をかけずに提供する】

- 適切に検査できたとしても、**不良品を多く生産してしまう**と利益は上がらない
- 品質の低い製品を生産しないよう**工程を監視**したり、**改善**したりする

工程の監視と改善も重要な品質管理活動

利益(りえき): profit

工程(こうてい)を監視(かんし)する: monitor the process

改善(かいぜん)する: improve

これをレポート作成に例えると

- ミスや不十分な部分(不良品に相当)がないか
提出前にチェックする
 - 体制の整備: **チェックリスト**を作ったり,
他人にチェックしてもらう
- レポート作成の工程管理
 - **原因**: **〆切直前に慌てて作成**している,
テキストや資料をよく読んでいない等
 - **改善**: 早めに始めるよう**スケジュール管理**,
大事なことはメモをとるようにする等

(C) 2007-2023 Hirohisa AMAN

6

工程管理(こうていかんり): production control

原因(げんいん): cause

〆切直前(しめきりちよくぜん): right before the deadline

慌てて(あわてて): in a hurry

よく読(よ)んでいない: did not read it carefully

改善(かいぜん): improvement

早(はや)めに始(はじ)める: begin earlier

スケジュール管理(スケジュールかんり): schedule management

ソフトウェアの場合

- 工場で作るような物理的な存在ではないが、概念としては同じことがいえる
- **検査**: 出荷前・リリース前に十分な**テスト**を行い、不具合があれば修正する
- **工程の監視と改善**: 開発工程における各種の**作業を記録・解析**し、その評価と改善を行う

物理的(ぶつりてき): physical

概念(がいねん): concept, notion

検査(けんさ): inspection

作業を記録(さぎょうをきろく): track the activity

解析(かいせき): analyze

評価(ひょうか): evaluation

作るだけでなく保守も大事

□ 保守とは、**ソフトウェアを適切に運用できるように維持**していく活動

- 運用開始後に見つかった障害の修正
- 要求の変化に対応するための修正
- 環境の変化に対応するための修正
- 継続的な品質の向上
- 障害発生の予防

.....

保守(ほしゅ): maintenance

運用開始後(うんようかいしご): after the beginning of the operation

障害(しょうがい): failure

修正(しゅうせい): modification, fixing

要求の変化(ようきゅうのへんか): change in the requirements

環境の変化(かんきょうのへんか): change in the environment/platform

継続的な(けいぞくてきな): continuous

予防(よぼう): prevention

保守の分類

ISO14764 – 2006

- ☐ 適応保守 (adaptive maintenance)
- ☐ 是正保守 (corrective maintenance)
- ☐ 緊急保守 (emergency maintenance)
- ☐ 改良保守 (maintenance enhancement)
- ☐ 完全化保守 (perfective maintenance)
- ☐ 予防保守 (preventive maintenance)

保守の分類(1/3)

□ 適応保守

環境の変化に対してソフトウェア製品を使い続けられるようにするために行う既存ソフトウェアの修正

例えば、OSやライブラリ、ハードウェアの変更等

□ 是正保守

運用開始後に発生した問題を訂正するために行う既存ソフトウェアの修正

つまり、バグの修正

レガシーシステムの対応は難しい
(例: COBOLのコード)

適応(てきおう)保守(ほしゅ): adaptive maintenance

是正(ぜせい)保守(ほしゅ): corrective maintenance

使い続ける(つかいつづける): continue to use it

既存(きぞん)ソフトウェアの修正(しゅうせい): modification of the existing software product

保守の分類(2／3)

□ 緊急保守

是正保守の一種であるが、**システム運用を確保するために**計画外で行う**一時的な**既存ソフトウェアの修正

急なトラブルが起こり、**ひとまずはその場をしのぐ**というもの

□ 改良保守

新しい要求を満たすための既存ソフトウェアの修正(機能追加を含む)

新たな機能を付け加えたり、**機能を統廃合**したりといった改良

緊急(きんきゅう)保守(ほしゅ): emergency maintenance
の一種(いっしゅ)である: a kind of ...

確保する(かくほする): secure

一時的な(いちじてきな): temporal

その場(ば)しのぎ: ad hoc

改良(かいりょう)保守(ほしゅ): maintenance enhancement

新たな機能(あらたなきのう): new functionality

統廃合(とうはいごう): elimination and consolidation

保守の分類(3／3)

□ 完全化保守

問題が起こっているわけではないが、将来のために行う既存ソフトウェアの修正

変更しやすいようにしておくとか性能を向上させておく等

□ 予防保守

障害として表面化する前に問題を発見し、それを是正するために行う既存ソフトウェアの修正

障害が起こってから対処するのではなく、それを未然に防ぐよう努める

完全化(かんぜんか)保守(ほしゅ): perfective maintenance

将来のために(しょうらいのために): for the future

変更しやすい(へんこうしやすい): easy to modify

性能(せいこう)を向上(こうじょう)させる: enhance the performance

予防(よぼう)保守(ほしゅ): preventive maintenance

表面化する前(ひょうめんかするまえ): before it arises

障害(しょうがい)を未然(みぜん)に防ぐ(ふせぐ): prevent a failure from occurring

【演習1】 是正保守と予防保守

- 是正保守と予防保守は, いずれもソフトウェアにおける問題箇所を修正するものである
- 両者の違いを説明せよ

是正保守(ぜせいほしゅ): corrective maintenance

予防保守(よぼうほしゅ): preventive maintenance

問題箇所(もんだいかしよ): problematic part

保守で必要なテスト： 回帰テスト

- 保守作業では、**修正を行うと同時に別のバグを作ってしまう**こともある
- 修正したモジュールだけでなく、それに**関係するモジュールもテストをやり直す**ことが大事：
これを**回帰テスト**という

1行のコードを変更しただけでシステムが暴走したり停止したりするかもしれない。回帰テストはきわめて重要である。

テストの記録を残しておくことが役立つ

回帰テスト(かいきテスト): regression testing

暴走(ぼうそう): runaway

停止(ていし): halt

テストの記録を残す(きろくをのこす): keep track of testing



品質管理の話題に戻ります

総合的品質管理 (Total Quality Control: TQC)

- 個人レベルではなく、経営者・リーダーをはじめ**企業・組織全員の参加と協力**で品質管理を効率的に実施すること

上に立つ人が品質管理を適切に理解・実践していなければ TQC はうまくいかない

- 多くの日本企業では TQC が当たり前
- 日本製品が高品質に評価されている要因の1つといわれている

(C) 2007-2023 Hirohisa AMAN

17

経営者(けいえいしゃ): manager
企業(きぎょう): company
組織(そしき): organization
参加(さんか): participation
協力(きょうりょく): cooperation
上に立つ人(うえにたつひと): boss
理解(りかい): understand
実践(じっせん): practice

日本での TQC 活動の特徴

1. 重点管理
2. 源流管理
3. 再発防止
4. データによる事実の把握
5. 標準化
6. PDCAサイクル

(1)重点管理

□ 現場で発生する問題に対して**大きな改善効果**が期待される項目を

■ **重点的に管理**

■ **優先順位付け**

■ 主な評価基準: **品質, コスト, 納期**

(**QCD: Quality, Cost, Delivery**)

重点管理(じゅうてんかんり): priority management

現場で(げんばで): in the field, on site

改善効果(かいぜんこうか): improvement effect

優先順位付け(ゆうせんじゅんいづけ): prioritization

評価基準(ひょうかきじゅん): criteria

(2)源流管理

- 問題の**発生源(最上流)**を特定して解決する
- 後の工程に影響を残さない

- **原因分析**が不可欠
- 場合によっては**体制の見直し**や新たな**投資**も必要

源流管理(げんりゅうかんり): upstream management

発生源(はっせいげん): source

最上流(さいじょうりゅう): upper stream

特定する(とくていする): detect, find

解決する(かいけつする): resolve

後の工程(あとのこうてい): following process

影響を残さない(えいきょうをのこさない): do not affect

原因分析(げんいんぶんせき): cause analysis

体制の見直し(たいせいのみなおし): reform of the organization/team

投資(とうし): investment

(3)再発防止

- 過去に起こってしまった問題を教訓にする
- 再発しないよう根本的な対策を講じる
 - なぜその問題が起こってしまったのかを客観的に分析する必要がある
 - 情報の共有とチェック体制も必要

再発防止(さいはつぼうし): prevention of recurrence

教訓(きょうくん): lesson

根本的な(こんぽんてきな): fundamental

対策を講じる(たいさくをこうじる): take measures

客観的に分析(きゃっかんてきにぶんせき): analyze objectively

情報の共有(じょうほうのきょうゆう): sharing of information

チェック体制(チェックたいせい): checking system

(4)データによる事実の把握

- 俗に**見える化**と呼ばれる概念
- 実際に起こっていることを**データ化**し, 問題を**論理的・客観的・統計的に**判断して処理する
- 対照的なのが KKD (経験, 勘, 度胸)
「よくわからないけどこれでうまくいっている」, 「多分これで大丈夫だと思う」, 「どうにかなるさ」

事実(じじつ): fact

把握(はあく): understanding

見える化(みえるか): visualization

論理的(ろんりてき): logically

客観的(きゃっかんてき): objectively

統計的(とうけいてき): statistically

判断する(はんだんする): judge, evaluate

対照的(たいしょうてき): contrasting

経験(けいけん): experience

勘(かん): intuition

度胸(どきょう): courage, guts

(5)標準化

- 作業内容についてその**標準**(standard)を設定し, それを活用すること
- いわゆる**マニュアル, ルール**を作る
 - うまくいく「型」を作り, それに従うことで**無用な誤り**を**防止**できる

標準化(ひょうじゅんか): standardization

無用な(むような): useless

誤りを防止(あやまりをぼうし): prevent errors

(6) PDCA サイクル

□ TQC を4つのフェーズのサイクルとして推進

- ① 計画を立てる(Plan)
- ② 計画に沿って実行する(Do)
- ③ 実行の結果を検討・評価・確認する(Check)
- ④ 良ければ維持するように努める(標準化等)
／悪ければ調整や再作業を行う(Action)

推進(すいしん): proceed

計画(けいかく): plan

実行(じっこう): execute

検討(けんとう): examine

評価(ひょうか): evaluate

確認(かくにん): check

維持(いじ): hold

調整(ちょうせい): adjust

再作業(さいさぎょう): rework

データ分析の例： パレート図(1／4)

□「所得の大部分は少数の人によるものである」 パレートの原理(Pareto principle)

■ もともとは経済の話で1896年にイタリアの経済学者 V.F.D. Pareto が発表

■ **80:20 の法則**(80-20 rule)とも呼ばれる
(例)

商品の売上の80%は, 20%の商品の販売による
会社の利益の80%は, 20%の社員の働きによる

所得(しょとく): income

大部分(だいぶぶん): majority

少数(しょうすう): fewer

売上(うりあげ): sale

利益(りえき): profit

データ分析の例： パレート図(2／4)

- 品質管理でも同様のことが言われている
(例)
 - 故障の80%は, 20%の部品に原因がある
 - バグの80%は, 20%のモジュールで見つかる

- 80%や20%という数字は正確ではないが,
そのような集中傾向にあるという意味

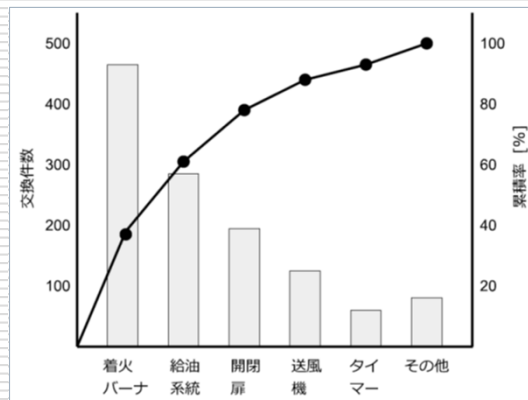
故障(こしょう): failure

部品(ぶひん): parts

原因(げんいん): cause

データ分析の例： パレート図(3/4)

- 項目を横軸として、件数の多い順に棒グラフを作り、あわせて累積グラフ(折れ線)も作る



これにより、**どこを改善するのが効率的なのか**が見えてくる

谷口博, 佐々木克彦, 早坂洋史, 小原伸哉, 谷口正行:
例題で学ぶ品質管理, 森北出版, 東京(2012).

(C) 2007-2023 Hirohisa AMAN

27

項目(こうもく): item

横軸(よこじく): horizontal axis, X-axis

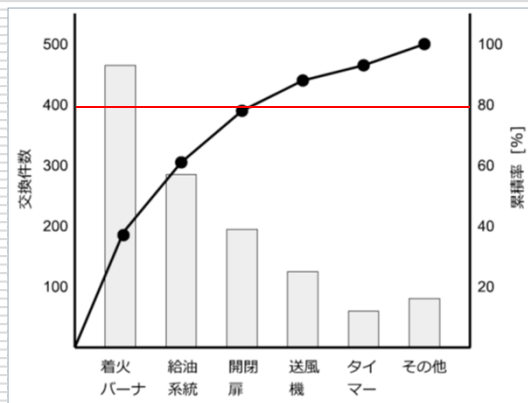
件数の多い順(けんすうのおおいじゅん): descending order of occurrences

棒グラフ(ぼうグラフ): bar chart

累積グラフ(るいせきグラフ): cumulative chart

折れ線(おれせん): line graph

データ分析の例： パレート図(4／4)



上位3つの部品

- 着火バーナ
- 給油系統
- 開閉扉

で全体の約80%を
占めているので、
これらの品質改善が重
要であることが分かる

着火バーナ(ちゃっかバーナ): ignition burner

給油系統(きゅうゆけいとう): oiling system

開閉扉(かいへいとびら): door



10-minutes rest break

10分休憩

品質管理の実例

【チェックシートを作る】

□ 目的: 作業の誤りや見落としを防止したり,
要件や基準について確認したりする

□ 作成のポイント:

- 外せない重要な(優先度の高い)事項を列挙
- **ありがちなミス**を列挙

実例(じつれい): actual example

チェックシート: check sheet

作業(さぎょう): work, activity

誤り(あやまり): error

見落とし(みおとし): oversight

要件(ようけん): requirement

基準(きじゅん): standard

優先度の高い(ゆうせんどのたかい): high priority

列挙(れっきょ): enumerate

ありがちな: frequent

ソフトウェアの品質保証

□ レビュー

開発活動の中に評価を組み込んでいく

- 要求仕様書のレビュー
 - 設計書のレビュー
 - ソースコードのレビュー
- その道の
熟練者・上級者
に見てもらう

□ テスト

網羅率といった基準を中心に確認していく

レビュー: review

開発活動(かいはつかつどう): development activities

評価(ひょうか): evaluation

要求仕様書(ようきゅうしょうしょ): requirements specification

設計書(せっけいしょ): design document

熟練者(じゅくれんしゃ), 上級者(じょうきゅうしゃ): skilled people

網羅率(もうらりつ): coverage

(例)プログラミングでの品質管理

- プログラムが長くなりすぎないように注意する
- プログラムが複雑化しないように注意する
- 他の人でもプログラムを理解できるようになっているかどうか確認する

長(なが)くなりすぎる: become too long

複雑化する(ふくざつかする): become complex

他の人(ほかのひと): other people

理解できる(りかいできる): comprehensible

(例1)プログラムの長さ

- 自分の書いた関数がどれくらいの長さになっているのか、その**行数を数えてみる**
- ただし、コメントや空行は数えない

- 長い関数があったら、それを分割すべきではないか考えてみる
 - **通常は長くても数十行程度がほとんど**

行数(ぎょうすう): line count

コメント: comment

空行(くうぎょう): blank line

分割(ぶんかつ): split, divide

数十行(すうじゅうぎょう): dozens of lines

(例2)プログラムの複雑さ

- 「複雑さ」という概念を単一の基準で測ることはできないが、あえて単純化すると
- 1つの関数の中で次の項目を数えてみる
 - 条件分岐(if や while, for 等)の個数
 - 変数の個数
 - 他の関数の呼び出し回数
 - 入れ子の最大の深さ

複雑さ(ふくざつさ): complexity

概念(がいねん): concept, notion

単一の基準(たんいつのきじゅん): single criterion

測る(はかる): measure

単純化(たんじゅんか): simplify

条件分岐(じょうけんぶんき): conditional branch

変数(へんすう): variable

呼び出し(よびだし): call

入れ子(いれこ): nest

最大の深さ(さいだいのふかさ): maximum depth

(例3)他の人でも理解できるか

□ 関数名, 変数名は**他人でも分かるような名前**になっているか

□ **インデント**は正しく付いているか

```
for ( i = 0; i < 100; i++ ){  
  min = i;  
  for ( j = i+1; j < 100; j++ ){  
    if ( a[min] > a[j] ){ min = j;  
  }  
}}
```



```
for ( i = 0; i < 100; i++ ){  
  min = i;  
  for ( j = i+1; j < 100; j++ ){  
    if ( a[min] > a[j] ){  
      min = j;  
    }  
  }  
}
```

□ 他人にチェック(レビュー)してもらったか

ソフトウェアメトリクス

- ソフトウェアの品質特性・特徴データを測定するための**定量的尺度**(その測定法の定義を含む)を**ソフトウェアメトリクス**(software metrics)

- ソフトウェアメトリクス(単に**メトリクス**と呼ぶ)の目的は、特性を**測定すること**であって、**評価値を与えることではない**ことに注意

品質特性(ひんしつとくせい): quality characteristics

特徴データ(とくちょうデータ): features

定量的尺度(ていりょうてきしゃくど): quantitative criteria

測定法(そくていほう): how to measure, how to quantify

評価値を与える(ひょうかちをあたえる): giving a score (mark, grade)

代表的なメトリクス①

□ Lines Of Code (LOC)

- **コード行数**のこと
- **プログラムの長さ(規模)**を測定するもの
- 定義にもよるが、**空行とコメント行は計上しないの**が一般的

- **KLOC** (=1000LOC) を単位としてソフトウェアシステムの規模を表したり、**バグ密度を 1KLOC あたりの平均バグ数**で表したりする

(C) 2007-2023 Hirohisa AMAN

37

長さ(ながさ): length

規模(きぼ): size

定義(ていぎ): definition

空行(くうぎょう): blank line

コメント行(コメントぎょう): comment line

計上しない(けいじょうしない): do not count

バグ密度(バグみつど): bug density

平均(へいきん): average

代表的なメトリクス②

□ McCabe メトリクス (サイクロマティック数: cyclomatic number)

- フローチャートでの制御フローの複雑さを測定
- プログラムの複雑さを表す
- フローチャートにおける独立な経路の数であるが、これは「条件分岐の個数+1」に等しい
 - サイクロマティック数 = 1 が、分岐の無いプログラムに対応し、これが最も単純な構造のプログラムと定義されていることになる

(C) 2007-2023 Hirohisa AMAN

38

制御フロー(せいぎょフロー): control flow

複雑さ(ふくざつさ): complexity

独立な経路(どくりつなけいろ): independent path

条件分岐(じょうけんぶんき): conditional branch

サイクロマティック数とテスト

- サイクロマティック数の大きなプログラムでは
実行パスが多い
- そのためホワイトボックステストの網羅率を高く
くするには、多くのテストケースが必要になる
- つまり、テストが難しいプログラムであることが多い

【演習2】

□ C プログラム `sample1202.c` について,
関数ごとの

□ LOC 値

□ サイクロマティック数

を答えなさい

lecture12-materials.zip をダウンロードしなさい
(その中に sample1202.c があります)

Answer the LOC value and the cyclomatic number of each function in sample1202.c

Download lecture12-materials.zip; it contains sample1202.c.

データ分析 (R を使った演習)

lecture12-materials.zip をダウンロードしなさい
(データファイル data12.csv,
R スクリプト Rscript12.R が含まれます)

データ分析(データぶんせき): data analysis

実際のデータを見てみる

□ 元データ

今回はデータの一部のみを使用

NASA IV&V Facility Metrics Data Program
プロジェクト CM1

<https://zenodo.org/>

- C で書かれた 20 KLOC の科学計測ソフトウェア
- モジュール数 505
- 40 種類のメトリクスデータ
(含 各モジュールでのバグ検出の有無)

(C) 2007-2023 Hirohisa AMAN

43

一部(いちぶ): a part of

科学計測(かがくけいそく): scientific measurement

バグ検出(バグけんしゅつ): bug detection

有無(うむ): presence or not

【R】CSV データの読み込み

- こちらで **data12.csv** を用意しています
- この内容を **cm1** という名前のデータフレームとして読み込みます

```
cm1 = read.csv( file.choose() )
```

【R】CSV データの読み込み・結果

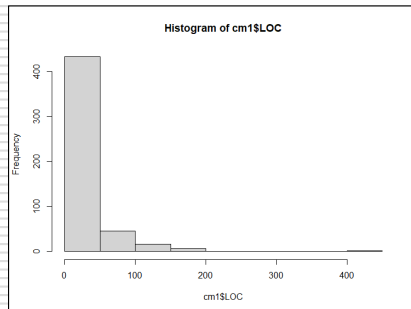
```
> cm1
  MODULE LOC  CC  BUG
1  24972  24   5   0
2  24973  31   4   1
3  24974  29   5   1
4  24975   7   2   0
5  24976  12   2   0
6  24977 106  13   0
7  24978  15   3   0
.....
```

列 名	意 味
MODULE	モジュールID
LOC	Lines Of Code (規模)
CC	サイクロマティック数 (複雑度)
BUG	バグの有無 ※有り:1, 無し:0

【R】LOC の ヒストグラムとカーネル密度推定

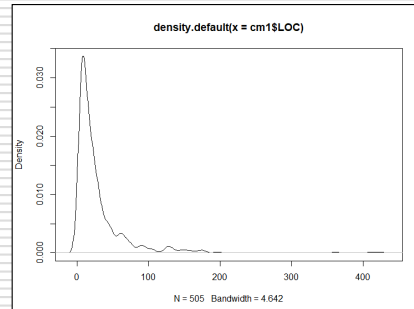
□ ヒストグラム

```
hist(cm1$LOC)
```



□ カーネル密度推定

```
plot(density(cm1$LOC))
```



(C) 2007-2023 Hirohisa AMAN

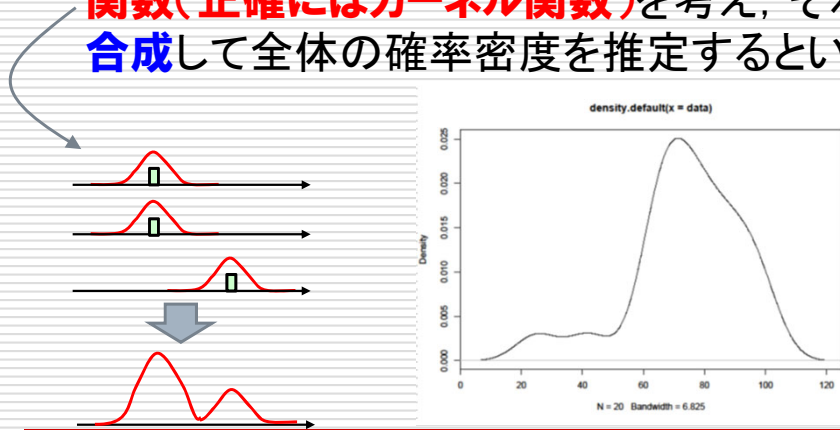
46

ヒストグラム: histogram

カーネル密度推定(カーネルみつどすいてい): kernel density estimation

(参考)カーネル密度推定

- 各データについて、それを中心とした**確率密度関数(正確にはカーネル関数)**を考え、それらを**合成**して全体の確率密度を推定するというもの



(C) 2007-2023 Hirohisa AMAN

47

参考(さんこう): reference

確率密度関数(かくりつみつどかんすう): probability density function

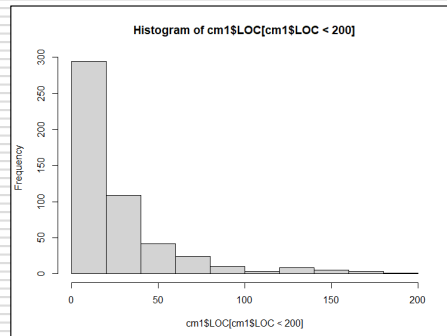
カーネル関数(カーネルかんすう): kernel function

合成(ごうせい): combine

【R】LOC のヒストグラム(続き)

□ LOC < 200 に限定して描き直してみると

```
hist(cm1$LOC[cm1$LOC < 200])
```



前ページのカーネル密度
推定からも予想できるが、
20 行未満のものが非常に多い傾向にあることが
分かる

(C) 2007-2023 Hirohisa AMAN

48

限定する(げんていする): limit to

20 行未満(20 ぎょうみまん): less than 20 lines

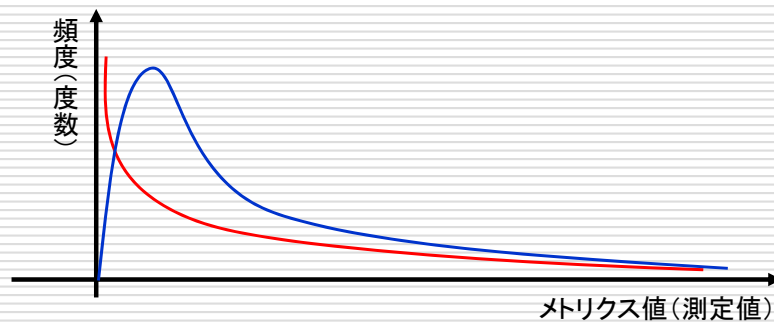
非常に多い(ひじょうにおおい): many

傾向(けいこう): tendency

メトリクス値の分布でよくある形

□ 「**右に歪んだ**」(right-skewed) 分布

※「右に裾が長い」(right-tailed) 分布ともいう



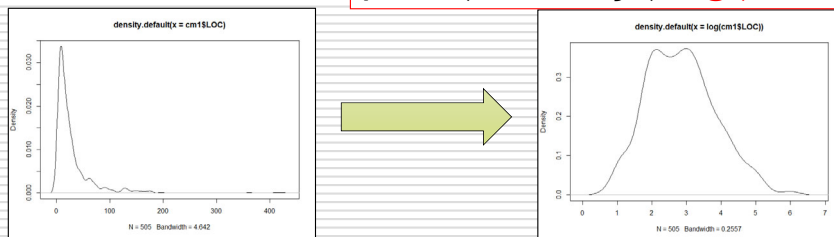
対数変換: $x \rightarrow \log x$

※ x に 0 が含まれる場合は +1 して $\log(x+1)$

□ **右に歪んだ分布**の場合, **対数変換**を施すことで分析しやすくなる場合がある

(例) 変換前では多くのデータが小さい範囲に集中しているが, 対数(log)をとることで分散化できる

```
plot(density(log(cm1$LOC)))
```



(C) 2007-2023 Hirohisa AMAN

50

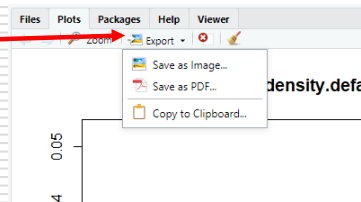
対数変換(たいすうへんかん): log transformation

分散化(ぶんさんか): decentralization

【演習3】(宿題:homework)

- サイクロマティック数(列名:CC)についても LOC と同様にヒストグラムの作成とカーネル密度推定を行いなさい
- ただし、いずれも**対数変換**を行うこと

「Plots」エリアの
「Export」メニューに
「Save as PDF」が
あるのでこれを使って
グラフを PDF として
保存して提出しなさい



Perform the log transformation for CC values and make PDF files of its histogram and kernel density estimation.

Submit those PDF files by tomorrow, 1pm.

You can save the graphs in PDF format by choosing “Export” → “Save as PDF” on Rstudio.

【R】 平均値と中央値

いずれも右に歪んだ分布
になっているため、**平均値
の方が中央値よりも大きい**

□ LOC の平均値と中央値

```
mean(cm1$LOC)  
median(cm1$LOC)
```



29.6099

16

□ サイクロマティック数の平均値と中央値

```
mean(cm1$CC)  
median(cm1$CC)
```



5.182178

3

平均値(へいきんち): mean

中央値(ちゅうおうち): median

【R】バグの有無で違いを見る

- **バグ有り**モジュールの LOC 値だけを取り出すには

```
cm1$LOC[ cm1$BUG==1 ]
```


BUG 列の値が 1 となっている場合のLOC 値だけを抽出
※ **cm1\$BUG==1** は TRUE, FALSE のベクトルとなる

- 同様に**バグの無い**モジュールだけを取り出すには **cm1\$BUG==0** と書く

【R】バグの有無で違いを見る(LOC)

□ **バグ有り**モジュールでの LOC 値の平均値と中央値

```
mean(cm1$LOC[cm1$BUG==1])  
median(cm1$LOC[cm1$BUG==1])
```




62.77083
42.5

明らかに差がある

□ 一方, **バグ無し**モジュールでは

```
mean(cm1$LOC[cm1$BUG==0])  
median(cm1$LOC[cm1$BUG==0])
```



26.12691
15

【演習4】(宿題:homework)

- バグ有りモジュールとバグ無しモジュールでの
サイクロマティック数に注目し,
その平均値と中央値を求めなさい

[12] Exercise-4
(〆切: 明日の 13 時)

Answer the mean values and the medians of the buggy modules and of the non-buggy ones.

Deadline: tomorrow, 1pm.

まとめ

- 保守の種類: 複数のタイプ; **回帰テスト**も重要
適応保守, 是正保守, 緊急保守, 改良保守,
完全化保守, 予防保守
- 品質管理
 - チェックシートは個人でも実践しやすい手法
 - **レビュー**, **テスト**の重要性
- ソフトウェアメトリクス
 - ソフトウェアの**特徴を数値化**
 - 評価にも使え, テストと関係が深い

(C) 2007-2023 Hirohisa AMAN

56

保守(ほしゅ)の種類(しゅるい): kind of maintenance

複数(ふくすう): two or more

回帰(かいき)テスト: regression testing

適応(てきおう)保守: adaptive maintenance)

是正(ぜせい)保守: corrective maintenance)

緊急(きんきゅう)保守: emergency maintenance)

改良(改良)保守: maintenance enhancement)

完全化(かんぜんか)保守: perfective maintenance)

予防(よぼう)保守: preventive maintenance)

品質管理(ひんしつかんり): quality management

個人(こじん): private individual

実践(じっせん)しやすい: easy to practice

特徴(とくちょう): character, feature

数値化(すうちか): quantification

テストと関係(かんけい)が深い(ふかい): tightly related to testing

宿題(homework)

“[12] Exercise-3” でPDFファイルを提出し,
“[12] Exercise-4” に答えなさい
(明日の 13 時まで)

Finish “[12] Exercise-3, Exercise-4”
by tomorrow 13:00 (1pm)

注意:スコアは final project の成績の一部となります
(Note: Your score will be a part of your final project evaluation)

【演習1】 是正保守と予防保守 (解答例)

- 是正保守は、ソフトウェアで起こってしまった問題を修正するものである
- 一方、予防保守は、問題が表面化する前に行う修正である
- つまり、問題が**表面化してから対処**するのが**是正保守**であり、**表面化する前に対処**するのが**予防保守**である

表面化(ひょうめんか)してから対処(たいしょ)する: deal with it after occurring
表面化する前に対処する: deal with it before occurring

【演習2】（解答例）

□ find_max

■ LOC = 10, サイクロマティック数 = 3

□ selection_sort

■ LOC = 9, サイクロマティック数 = 2

□ main

■ LOC = 21, サイクロマティック数 = 4