

ソフトウェア工学

第5回

土田 隼之

週	授業内容・方法	週ごとの到達目標
1週	ソフトウェアの性質と開発	ソフトウェア開発の特徴および課題について少なくとも一つ上げられ、その理由を言える。
2週	ソフトウェア開発プロセス	複数の開発プロセスモデルを挙げ、それぞれの特徴を言える。
3週	要求分析	要求分析とプロトタイピングの関係性や有用性について言える。
4週	ソフトウェア設計	モジュールの結合度の低い場合と高い場合のモジュール間の依存性について述べ、モジュール結合度の低い具体例を言える。
5週	プログラミングとテスト	誤り混入をさせないためのプログラミング手法およびテスト効率を向上させる技法について言える。
6週	テストと保守	保守容易性を確保するための方策について、考察し、述べることができる。
7週	グループワーク	前半6週に関する課題を、グループワークで取り組む。
8週	中間試験	前半に習得した項目について確認する。
9週	オブジェクト指向 1	身の回り
10週	オブジェクト指向 2	オブジェ
11週	ソフトウェア再利用	ソフトウ
12週	プロジェクト管理	プロジェ
13週	品質管理	品質管理
14週	ソフトウェア開発規模と見積もり	ソフトウェア開発規模の見積もり手法について言える。
15週	グループワーク	後半6週に関する課題を、グループワークで取り組む。
16週	期末試験	後半に習得した項目について確認する。

中間試験の前には、模擬問題を予定

→傾向を見てもらう目的。ポートフォリオ点に入るので、試験点には影響なし。

#課題をきちんと出していれば、模擬試験0点でもポートフォリオ点満点

「語句と意味の紐づけ」などの問題を出題予定

中間試験の時期についてのアンケート

<https://forms.office.com/r/U4ebZEwEGP>

ソフトウェア工学の中間試験の時期について

1. ソフトウェア工学の中間試験の時期は、いつくらいが良いですか？都合の良い時期を全て選択してください。 *

☐ シラバス通りの第八週(6/9)

☐ 第九週(6/16)

☐ 第十週(6/23)

今日の内容

1)プログラミングと開発プロセス

2)内部設計の前回残り

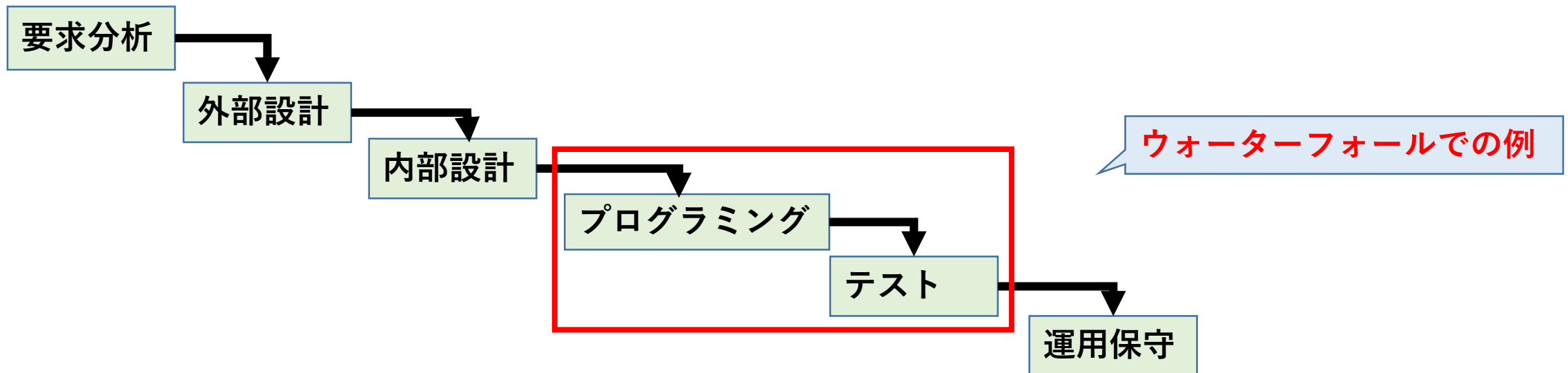
3)コーディング概要

4)プログラミング書法

1) プログラミングと開発プロセス

「内部設計書」をもとに、実装する。

#内部設計書には、例えば1)処理全体の処理フロー(フローチャートなどで記載)と2)各モジュールの引数や返り値、3)各モジュールの処理フローが記載されている。



今日の内容

1) プログラミングと開発プロセス

2) 内部設計の前回残り

3) コーディング概要

4) プログラミング書法

2)内部設計の例

SQL文を実行するための主な機能として下記がある。

- (1)SQL文解析:受信SQL文を解析し、使われているSQL句を明らかにする。
- (2)SQL文最適化:解析結果から、処理量減のため受信SQL文の変更やデータアクセスに索引を使うかの判断などを行い、クエリ実行プランを作成する。
- (3)クエリ実行エンジン:上記で作成されたクエリを実行する。

```
select *  
from tableA,tableB,tableC  
where tableA.a1=tableB.b1  
and tableB.b2=tableC.c2
```



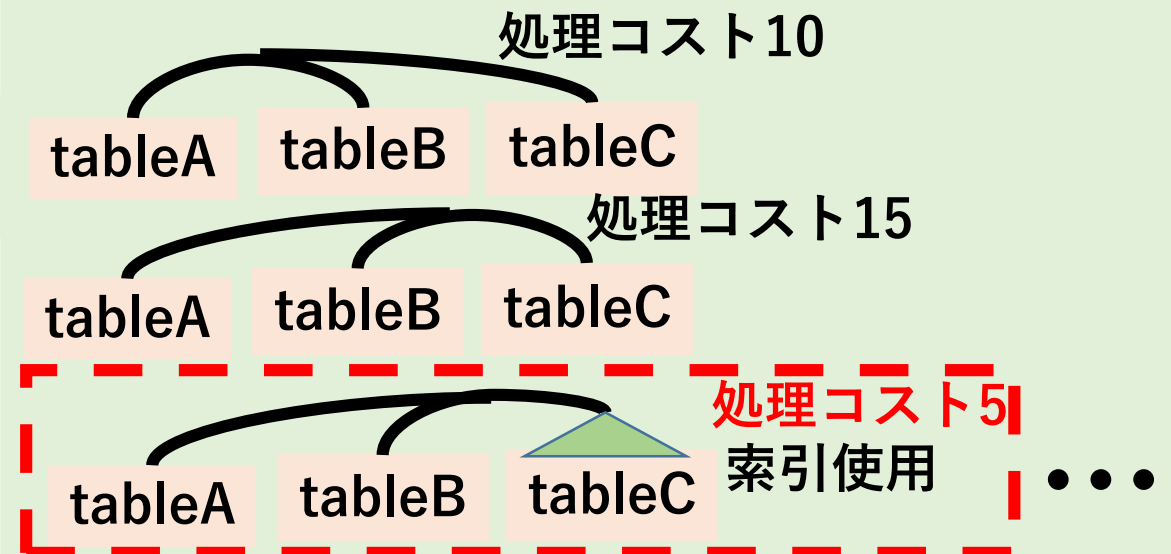
データベース

SQLの解析

SQLの最適化

クエリ実行
エンジン

抽出カラム: すべて
操作対象表: tableA, tableB, tableC
結合条件: tableA.a1=tableB.b1
tableB.b2=tableC.c2



2) 内部設計の例

mysql / mysql-server Public

<> Code Pull requests 6 Actions Projects Security Insights

8.0 14 branches 719 tags Go to file Add file Code

mysql-builder@oracle.com No commit message 3290a66 16 days ago 168,445 commits

Docs	Bug #33217957 Update howto for building with clang on Windows.	2 months ago
client	WL#11102: MySQL OCI IAM authentication	2 months ago
cmake	Bug #33388069: ABI INCOMPATIBILITY after adding new member to	29 days ago
components	Improve the error log messages of the backup_page_tracker.	2 months ago
doxygen_resources	BUG#32368342: UPDATE COPYRIGHT HEADERS TO 2021	9 months ago
sql-common	41: Redesign the communication of the 2nd and 3d MFA fact...	2 months ago
sql	Revert "Bug#32427727: Functional index ignored by SELECT query fro..."	last month
storage	BUG#33290335 INPLACE DDL leaves pages with dangling flush obser...	24 days ago
strings	Bug #32992125: USE C++17 [[FALLTHROUGH]] ATTRIBUTE	4 months ago
support-files	BUG#32368342: UPDATE COPYRIGHT HEADERS TO 2021	9 months ago

SQL解析処理

2) 内部設計の例

mysql / mysql-server Public

<> Code Pull requests 6 Actions Projects Security Insights

8.0 mysql-server / sql / Go to file

roylyseng and dahlertend Revert "Bug#32427727: Functional index ignored by SELECT query from w... d7a6bf9 on

..	
auth	BUG#31716706 BAD HANDLING OF QUOTES IN SHOW GRANTS
binlog	BUG#32368342: UPDATE COPYRIGHT HEADERS TO 2021
changestreams/apply	WL#7491: GTID-based replication applier recovery and positioning
conn_handler	Bug #32907475: USE [[MAYBE_UNUSED]]
containers	BUG#32368342: UPDATE COPYRIGHT HEADERS TO 2021
daemon_proxy_keyring	WL#13446: Migrate Keyring APIs to Components
dd	Bug#33002479 "HIGH MEMORY ALLOCATION FOR DROP DATABASE" [noclose]
examples	2021
gis	HEAP WITHOUT FREEING IT
histograms	bug #33025668: REMOVE INIT_ALLOC_ROOT()
join_optimizer	Bug#33108575: HYPERGRAPH - INVALID READ IN FILTERITERATOR::READ()
locks	Merged BUG#32368342 from 5.7 to 8.0.
memory	Bug #33004840 CLEAN UP C++ CODE FOR SOLARIS
partitioning	Bug #32907475: USE [[MAYBE_UNUSED]]

SQL最適化(ジョイン最適化)

<https://github.com/mysql/mysql-server/tree/8.0/sql>

2)モジュール分割の評価

モジュール分割の良否は、設計の良否に直接影響するため非常に重要である。関連項目として、下記がある。

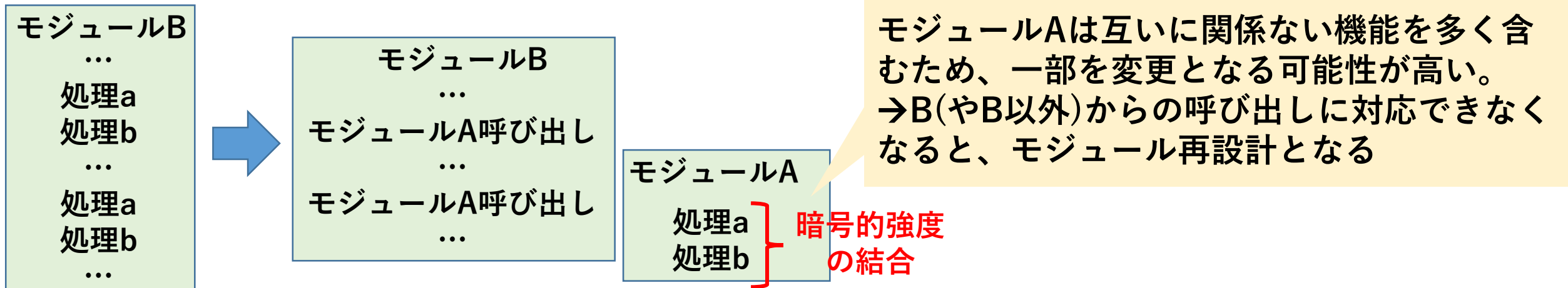
- 1)モジュールの大きさ:**人間の認知範囲を反映させることができる程度の大きさにするとよい。例えば、プログラム100行以内、ディスプレイ2画面程度で見ることができる程度など。
- 2)情報隠蔽の度合:**モジュールの使用とは関係ないモジュール内部の設計事項を隠すこと。モジュール使用者は、不要な情報を理解する必要がなくなり、修正時も他モジュールへの影響度が小さくなる。
- 3)モジュール強度:**モジュール内の命令群が深く関わりあっているかの強さ。
- 4)モジュール間結合度:**2つのモジュール間の結合の強さをモジュール間結合度といい、低いほど良い設計といわれる。

2) モジュール強度1

- 1つのモジュール内の機能間の関連性をモジュール強度という。レベルが大きい(数字が大きい)ほど強く、良いモジュールとなる。モジュール強度があまりに弱いときは、さらなるモジュール分割を吟味する必要がある。

1) 暗号的強度:モジュール間で機能の関連がない場合。お互いに全く関係なく複数の機能を実行している。暗号とは「偶然に物事が一致する」という意味である。

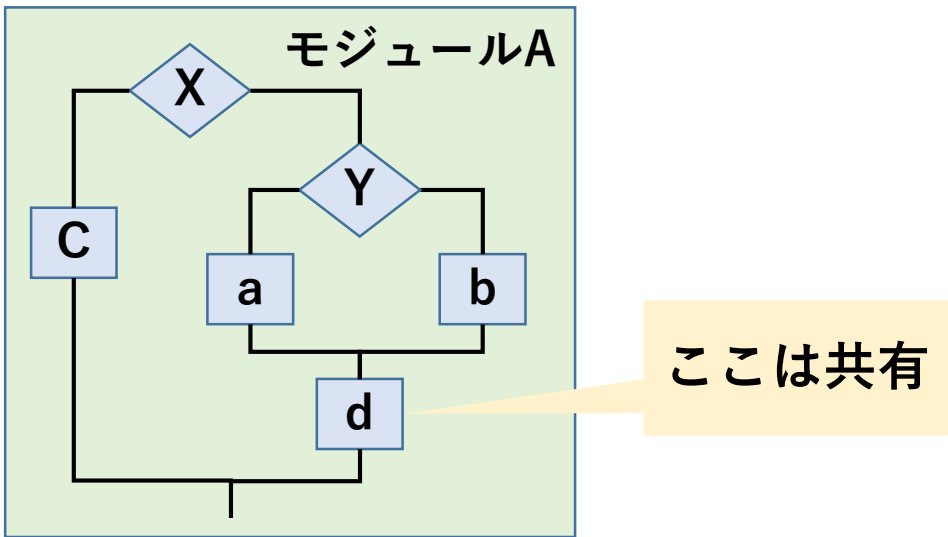
例えば、たまたまモジュール内に重複している命令群のパターンがあったので、結合して1つのモジュールにした場合である。この場合、複数の全く関係ない機能进行处理するため、モジュール機能を正しく命名できない。



2) モジュール強度2

2) 論理的強度: 関連する複数の機能を持つモジュールで、制御変数により機能を選択する。例: すべての入出力操作をまとめてモジュール化

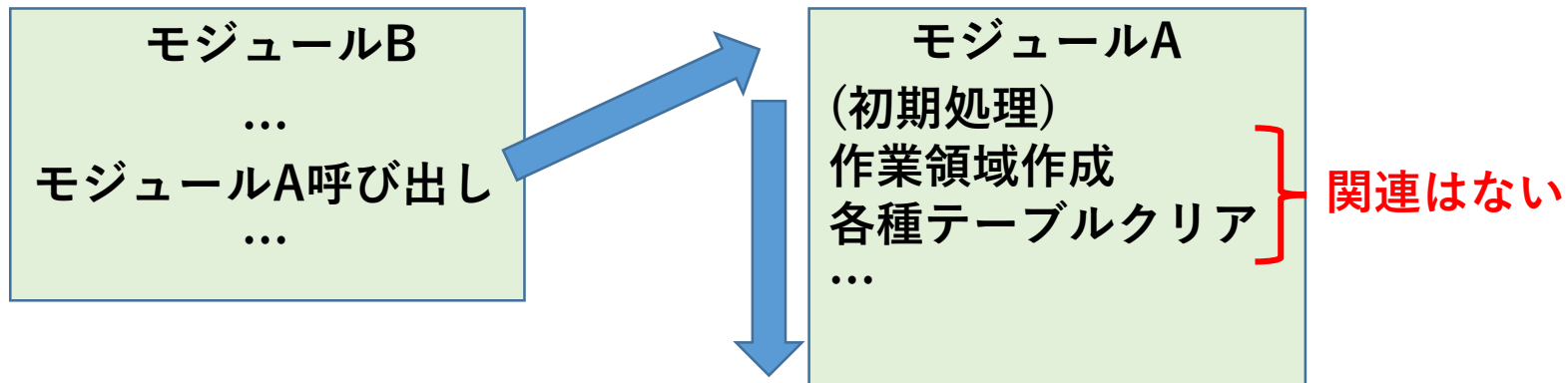
モジュール内の論理は、条件によって実行パスが異なることとなる。つまり、論理的強度のモジュールは、関連したいくつかの機能を含み、そのうちの1つだけが呼び出しモジュールによって(論理)識別され、実行されるモジュールである。呼び出されたとき、実行される命令は一部だけであり、モジュール内のすべての命令が実行されるわけではないため、内包される命令群の関連性は弱く、モジュール強度は弱くなる。



2) モジュール強度3

3) 時間的強度: 機能要素間の関連は薄いですが、一群の機能要素がある時間的順序関係に沿って起動される。機能間にあまり強い関連性はないが、特定の時点に連続して実行される。

代表的なものとして、「初期処理モジュール」がある。ソフトウェアの初期処理では、作業領域作成や各種テーブルのクリアを、それらの処理に強い関連がなくても、実行の最初にまとめて行われる。

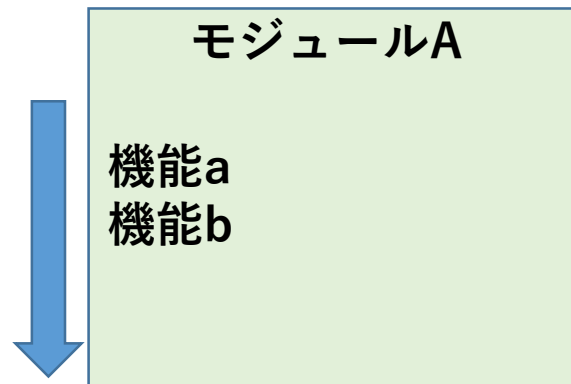
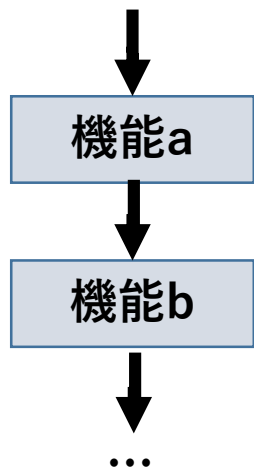


2) モジュール強度4

4) 手順的強度: モジュールを構成する各機能の起動には順序関係があり、それに従いまとめられている。具体的には、問題进行处理のために関係している複数個の機能のうちの、いくつかを実行する。実行される複数機能は順番(手順)に実行される。基本的に、時間的強度の特性を持つが、機能間に手順的な関連性がある分、時間的強度より強度は強くなる。

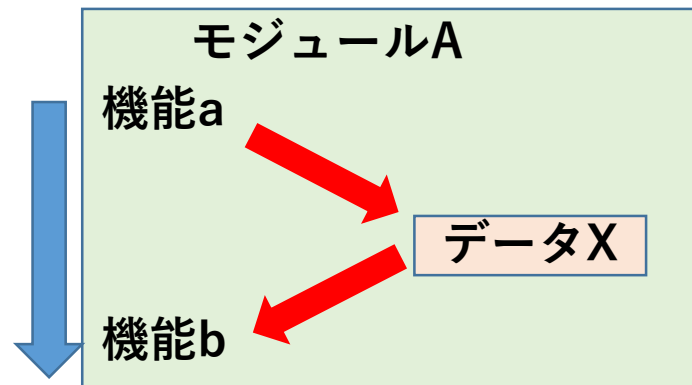
大きな機能の一部の手順を1つのモジュールにした場合、手順強度となる。これは、フローチャートの一部をモジュール化した場合などに相当する。

フローチャート



2) モジュール強度5

5) 連絡的強度: 各機能の起動に順序があり、かつ各機能が共通のデータを参照または変更する。基本的に、手順的強度の特性を持つ。手順的強度と異なるのは、モジュール内機能間でデータの受け渡し(連絡)をしたり、同じデータを参照する点である。モジュール内機能がデータに関してつながっている分、手順的強度より強度が高くなる。

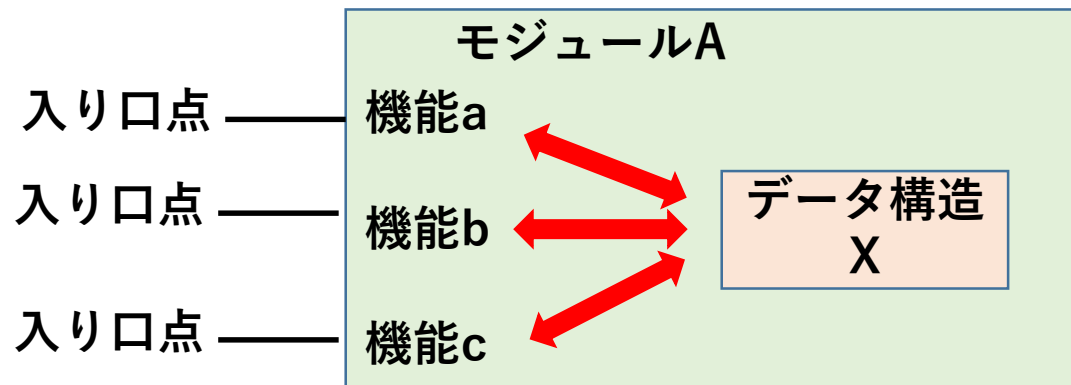


2) モジュール強度6

6) 機能的強度: 単一機能として考えることができる機能群からなる1つのモジュールで機能を1つ実現している場合である。具体的には、特定のデータ構造を扱う複数の機能を1つのモジュールにまとめたものである。

同一データ構造(情報)は、できるだけ特定のモジュールだけでアクセスしようという発想である。このようにしておけば、データの修正時の影響をこのモジュールだけに限定でき、情報隠蔽の面からも有利となる。

論理的強度も、情報隠蔽の特徴をもっているが、入口点の個数が異なる。



2) モジュール強度7

7) 情報的強度: 同じ内部データを扱う機能を集め、かつ1つのモジュールの機能は単一の場合。オブジェクト指向のカプセル化の概念に近い。具体的には、モジュール内のすべての命令が1つの役割(機能)を実行するために関連しあっているモジュールである。

モジュールA

Matrix Add(x,y){

...

(行列加算関連処理のみ)

...

}

課題5-1モジュール強度(締切:6/2)

作成したユースケースにおいて作成が見込まれる関数について

モジュール強度が異なる関数2つを記載(モジュール概要と引数、返り値)せよ。そして、なぜその関数とその強度となるかを説明せよ。

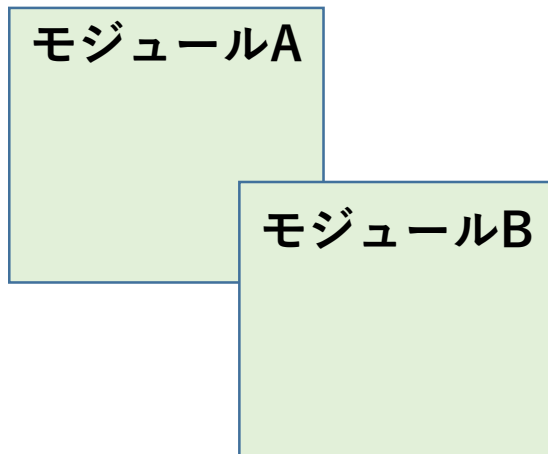
#モジュール強度は、そのモジュールで可能なもっとも高いものとせよ

例えば、暗号的強度のモジュールはどんな場合でも可能であるが、より強度を高く出来る場合には、もっとも強度が高いレベルとせよ。

2) モジュール間結合度1

モジュール間結合度は、2つのモジュールの間の結合が密である程度を示す。モジュール間結合度の定義として下記がある。結合度は数字が高いほど関係が薄く、良いモジュールとされる。

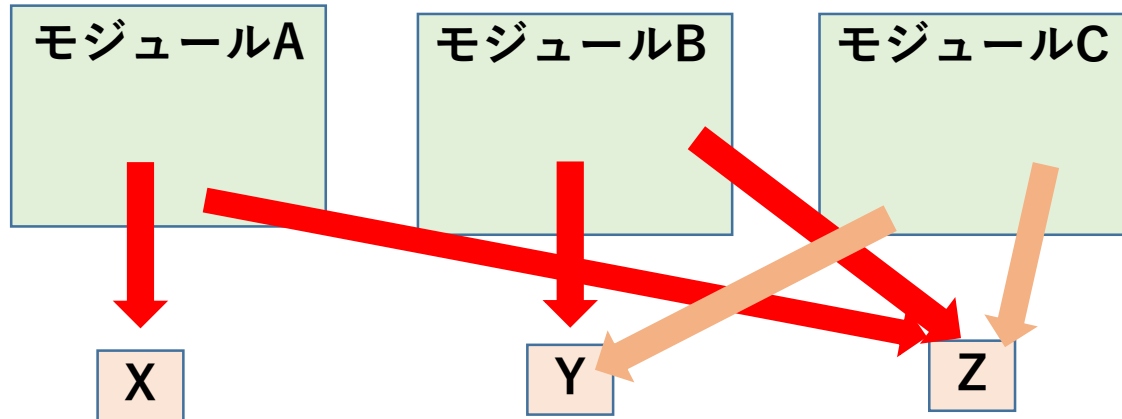
1) 内容結合: あるモジュールが、他のモジュール内の構成要素を直接参照したり、変更する結合をいう。例えば、GOTO文で直接他のモジュールに移動する場合である。



2) モジュール間結合度2

2) 共通結合: 複数のモジュール間で、ある共通領域(例: グローバル変数)を参照している結合をいう。1つのモジュールがデータ領域を変更すると他のモジュールに影響を与える。デメリットが多くある。

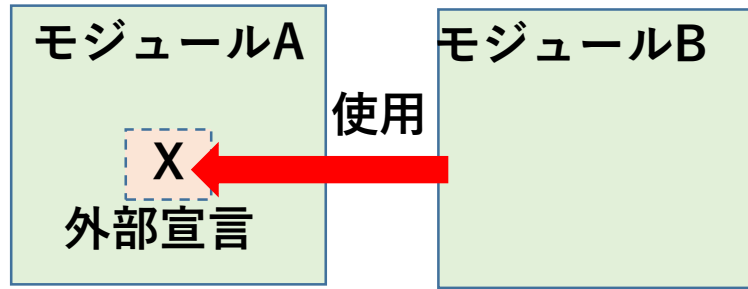
- ・ 共有領域のデータは、モジュール間のインタフェース上に現れないので、コード読解を難しくする。
- ・ 共通領域のデータは、もともとそのデータに無関係なモジュールでも、その気になれば利用できてしまうので、コード安全性が低下する。
- ・ 共通結合しているモジュールは、共通領域のデータを通じて色々なモジュールとつながってしまっているため、再利用性も阻害する。



2) モジュール間結合度3

3) 外部結合: あるモジュール内で外部参照可能であると宣言されたデータ領域を、他のモジュールが直接参照する結合をいう。外部参照可能であるとは、例えばpublic宣言された変数のことである。

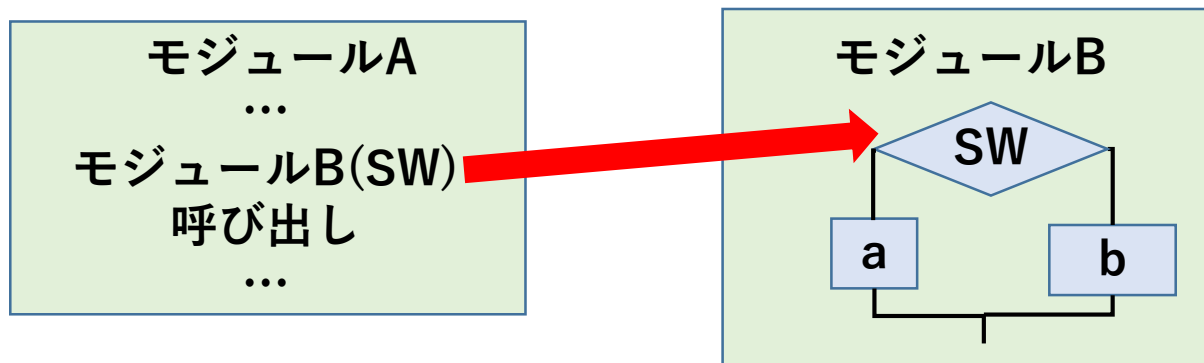
データ共有という意味では共通結合と似ているが、必要なデータだけを外部宣言するため、共通結合のように、本来不必要なデータまで共有することがないため、共通結合より結合度は弱くなる。



2) モジュール間結合度4

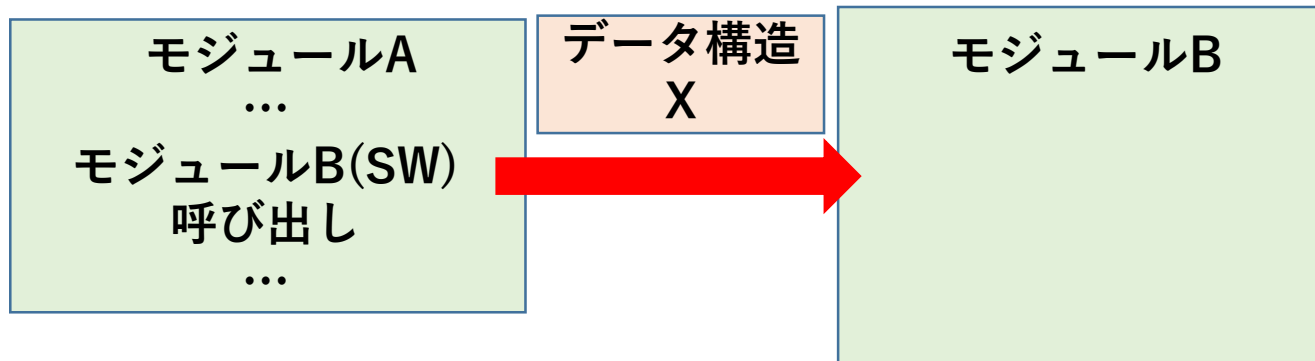
4)制御結合:あるモジュールが他のモジュールを呼び出す時、制御のためのフラグやパラメータを渡す。呼び出すモジュールは、呼び出されるモジュール内の制御を理解している。

制御結合では、パラメータの一つとしてスイッチ変数を渡し、呼び出されるモジュールがその時に行う機能を指示する。そのため、呼び出し側は、呼び出されるモジュールの論理を知っている必要があり、相手をブラックボックス扱いにできず、結合度が強くなる。ただ、データ共有ということではないので、共通結合や外部結合より結合度は弱いといえる。



2) モジュール間結合度5

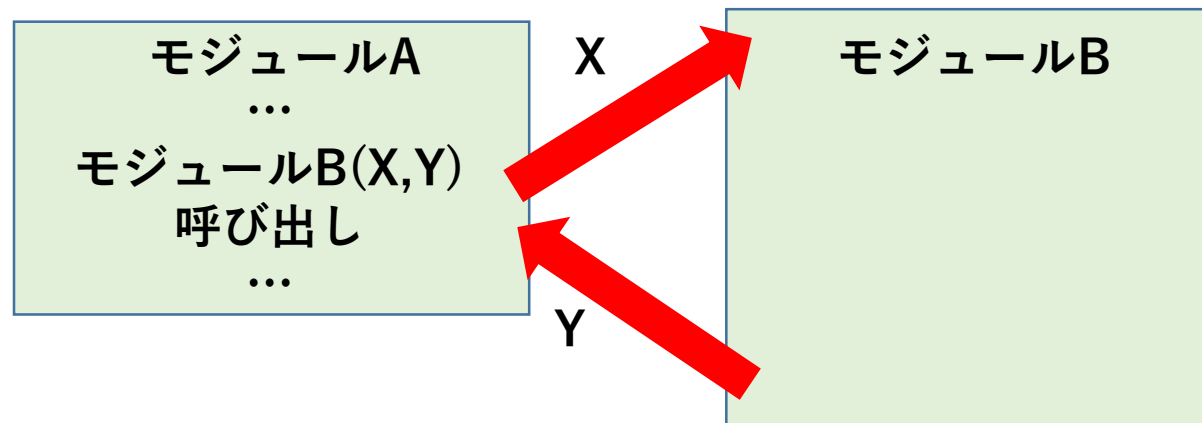
5) スタンプ結合: 共有データ領域にはない同じ構造のデータ(構造体)を受け渡す結合をいう。構造内の(データ順序と)データ型をモジュール間で知っている必要がある。一方の構造体の仕様を変更すると、他方も修正が必要になる。スタンプ結合の場合、受け渡すデータ構造の一部を使用しないことがある。



2) モジュール間結合度6

6) データ結合: データの必要な部分だけを、(構造体でなく) スカラ型で呼出しモジュールへの引数として渡す結合をいう。呼び出されるモジュールと呼び出すモジュールのデータ構造に関する特別な知識は必要ない。

相手モジュールをブラックボックス化できるので、結合度は一番弱くなる。モジュール間の結合は、明確化されたパラメータでデータを受け渡す、データ結合が一番良いとされている。



課題5-2モジュール結合度(締切:6/2)

作成したユースケースにおいて作成が見込まれる関数について

モジュール結合度が異なる関数の関係2つを記載(モジュール概要と引数、返り値)せよ。そして、なぜその関数とその結合度となるかを説明せよ。

#モジュール結合度は、そのモジュールで可能なもっとも高いものとせよ

例えば、内容結合にすることは、どんな場合でも可能であるが、より結合度を高く出来る場合には、もっとも結合度が高いレベルとせよ。

今日の内容

1)プログラミングと開発プロセス

2)内部設計の前回残り

3)コーディング概要

4)プログラミング書法

3)コーディング概要

内部設計書の内容に沿ってそれを実現するプログラムを書き、記述内容をチェックし、内部設計書通りに動作するかどうかをテストなどで確認する。

プログラミング実装は、一般的に以下の手順で進められる。

- 1)コーディング
- 2)ソースコードレビュー
- 3)単体テスト

3) コーディング

ソースコードの作成を行う作業である。内部設計書の内容に沿ってプログラミング言語を使って、ソースコードを書きます。

関数やプログラムの名前、受け取るデータの数と種類、内部で行うべき処理、呼び出し元へ返す処理結果は内部設計書で定義されています。プログラマは、内部設計書と合致するように作成する必要があります。

3) ソースコードレビュー

関数などの最小単位のプログラムを書き終え、コンパイルエラーがないことが確認出来たら、レビューを行います。ソースコードレビューでは、参加者全員でソースコードを読み、以下のような観点で内容をチェックします。

- 1) コーディング規約に沿って書いてあるかどうか
- 2) 内部設計書と合致する正しいロジックで書いてあるかどうか

コーディング規約: プログラマ全員が守るべき、ソースコードの書き方の作法を定めたもの。ソースコードは芸術作品ではないので、書き方の個性は一般的に必要ない。

3) コーディング規約

日本語で書かれた文章と同様に、ソースコードにおいても表記方法は統一されている必要がある。例:「ですます調」と「である調」の混在、ひらがなとカタカナの使い方が変(例:かたカナ)など。

ソースコードの場合、名前の付け方などがプログラム毎に異なると、読みにくかったり、読み間違える可能性が高くなったりする。そのため、コーディングする前に、記述ルールをコーディング規約として定める。複数のプログラマで手分けしてソースコードを書く場合でも、記述形式が統一され、保守性が高まることとなる。

コーディング規約の例:

- ・プログラムの冒頭に付けるコメントの記述ルール
- ・変数名の付け方や宣言方法に関するルール

例:変数名はその変数の持つ意味を英語で記述する。英字は小文字を使用するものとし、複数の英単語を連結する場合に限って、2単語目以降の各単語の先頭文字を大文字とする。

3)単体テスト

ソースコードレビュー終了後に、単体テストを行う。単体テストは、関数などのプログラム単体の品質を確保するのが目的である。

レビューが終わり、記述上はロジックに問題がないと判断できたプログラムを実際に動作させ、内部設計書通りに動作しているかを確認する。

人間の検出能力には限界があるため、単体テストが必要となる。

3) プログラミングと実際の業務

• プログラムは千差万別

例: 公共システムのUI

例: パッケージソフト(データベース, OS)

単純作業寄りのプログラム、職人芸が必要な(ややこしい)プログラム、
#製品では、不必要に職人芸(わかりにくい)なプログラムをするのは厳禁
→ 自分以外の人、自分が書いたプログラムを保守できる必要

製品は10年単位で使われる可能性がある。バグが出ると、誰かが直す。

開発メンバは良くも悪くも入れ替わる。海外駐在とか、製品を使った企画に異動するかも

納期に遅れると大問題
スケジュール=人件費

• プログラミング(=システム開発)はスケジュール命

→ 周りの人と調整をしながら仕事を進める。

作業分担は自分で負荷量を適切に調整。居る意味ないのも良くない
#責任感は必須だが、負荷大は。。

まずは、どんなプログラムを作るのか資料を作成

社外に出るような製品ではいきなりコーディングすることは無い

資料が無いと、作業量などがわからないので、スケジュールが作れない

3)プログラミングと資料

- ・システム開発(プログラミング)はチーム(例:100人,20人)で行う
どんな開発でも、子会社や外注のメンバと一緒に開発する。

本体社員は儲かるソフト案を作るのが仕事。

仕様決まれば、(たいていの場合は)実装は誰でも出来るという位置付け

- 様々な理由(仕事量の増減に対応、コスト削減、管理職ポスト)
プログラムの仕様作成は本体で行う。他人がわかる実装資料を作る必要
すごく仕事の出来る外注先もあるし、そうでない場合も

3)職人芸なプログラミングの例

例えば、プログラム実行中にどれくらいCPUが効率的に処理しているかのデータが取れるツール(プロファイラ)で状況を確認する。

perfの使い方(イベントカウント 2/2)

15/58

使い方 (イベント指定)

```
$ perf stat -e cache-misses,cache-references ./a.out
```

キャッシュミス回数 キャッシュの参照回数

出力

Performance counter stats for './a.out':

```
1,019,158 cache-misses      # 5.927 % of all cache refs
17,194,391 cache-references
```

0.444152327 seconds time elapsed

一般的には、CPUネックだとキャッシュミスが多い→キャッシュミスを減らす

17,194,391回キャッシュを参照にあって、そのうち
1,019,158回キャッシュミスしたから、
キャッシュミス率は5.927%だよ、という意味



ISSP, Univ. of Tokyo

3)職人芸なプログラミングの例

CPU(例:xeon,core i7)は、CPU内の特定処理が起こった回数を記録している場合がある。その場合、その情報をプロファイラで取得できる。

プロファイラ(イベント取得型)

13/58

Hardware Counter

CPUがイベントをカウントする機能を持っている時に使える

取得可能な主なイベント:

- ・実行命令 (MIPS)
- ・浮動小数点演算 (FLOPS) ←こういうのに気を取られがちだが
- ・サイクルあたりの実行命令数 (IPC)
- ・キャッシュミス
- ・バリア待ち ←個人的にはこっちが重要だと思う
- ・演算待ち

一般的には、CPUネックだとキャッシュミスが多い→キャッシュミスを減らす

プロファイラの使い方

システム依存

Linux では perfを使うのが便利

京では、カウントするイベントの種類を指定

カウンタにより取れるイベントが決まっている

→同じカウンタに割り当てられたイベントが知りたい場合、複数回実行する必要がある



3)職人芸なプログラミングの例

インテルのドキュメントなどを確認して、例えば、プリフェッチ(キャッシュにデータを持ってきてくれる)をやる。

Overview

Combined Volume Set of Intel® 64 and IA-32

Architectures Software Developer's Manuals

Four-Volume Set of Intel® 64 and IA-32

Architectures Software Developer's Manuals

Ten-Volume Set of Intel® 64 and IA-32

Architectures Software Developer's

Manuals

Intel® Architecture Instruction Set

Extensions Programming Reference

Software Optimization Reference Manual

Public Repository on GitHub

Uncore Performance Monitoring Reference
Manuals

Related Specifications, Application Notes,
and White Papers

Last Updated: 10/25/2021

Ten-Volume Set of Intel® 64 and IA-32

Architectures Software Developer's Manuals

This set contains the same information as the four-volume set, but separated into ten smaller PDFs: volume 1, volume 2A, volume 2B, volume 2C, volume 2D, volume 3C, volume 3D, and volume 4. This set is better suited to those who

Document	Description
Intel® 64 and IA-32 architectures software developer's manual volume 1: Basic architecture	Describes the architecture of Intel® 64 and IA-32 architectures.
Intel® 64 and IA-32 architectures software developer's manual volume 2A: Instruction set reference, A-L	Describes the format of instructions (from A to L). This volume covers instructions A through L.
Intel® 64 and IA-32 architectures software developer's manual volume 2B: Instruction set reference, M-U	Provides reference information for instructions M through U.

Intel® 64 and IA-32 Architectures Software Developer's Manual

257 / 482

—

100%

+

🔍

🔄

destination region should also be mapped as WC. If mapped as WB or WT, there is the potential for speculative processor reads to bring the data into the caches; in this case, non-temporal stores would then update in place, and data would not be flushed from the processor by a subsequent fencing operation.

10.4.6.3 PREFETCHh Instructions

The **PREFETCHh** instructions permit programs to load data into the processor at a suggested cache level, so that data is closer to the processor's load and store unit when it is needed. These instructions fetch 32 aligned bytes (or more, depending on the implementation) containing the addressed byte to a location in the cache hierarchy specified by the temporal locality hint (see Table 10-1). In this table, the first-level cache is closest to the processor and second-level cache is farther away from the processor than the first-level cache. The hints specify a prefetch of either temporal or non-temporal data (see Section 10.4.6.2, "Caching of Temporal vs. Non-Temporal Data"). Subsequent accesses to temporal data are treated like normal accesses, while those to non-temporal data will continue to minimize cache pollution. If the data is already present at a level of the cache hierarchy that is closer to the processor, the PREFETCHh instruction will not result in any data movement. The PREFETCHh instructions do not affect functional behavior of the program.

See Section 11.6.13, "Cacheability Hint Instructions," for additional information about the PREFETCHh instructions.

Table 10-1. PREFETCHh Instructions Caching Hints

PREFETCHh Instruction Mnemonic	Actions
PREFETCHT0	Temporal data—fetch data into all levels of cache hierarchy: <ul style="list-style-type: none">• Pentium III processor—1st-level cache or 2nd-level cache• Pentium 4 and Intel Xeon processor—2nd-level cache
PREFETCHT1	Temporal data—fetch data into level 2 cache and higher <ul style="list-style-type: none">• Pentium III processor—2nd-level cache• Pentium 4 and Intel Xeon processor—2nd-level cache

メーカーの研究職だと、こういう報告を毎週やる。必ずしも毎回進捗無くても良いが、5回に4回はそれなりの進捗が必要
#自分で実装して性能評価も必要

<https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html>

3)職人芸なプログラミングの例

CPU構造は、ヘネパタ(作者がヘネシーさんと、パターソンさん)などを読んで確認する。



2点すべてのイメージを見る

コンピュータアーキテクチャ[第6版]定量的アプローチ 新書 - 2019/9/25

ジョン・L・ヘネシー (著), デイビッド・A・パターソン (著), & 4 その他

★★★★☆ ~ 9個の評価

すべての形式と版を表示

新書
¥8,800
獲得ポイント: 88pt

¥8,200 より 13 中古品

¥8,800 より 14 新品

チューリング賞受賞(2017年度)記念翻訳版

コンピュータ科学最高の栄誉であるチューリング賞をRISCアーキテクチャへの貢献により2018年に受賞したヘネシーとパターソンによる

『コンピュータアーキテクチャ 定量的アプローチ[第6版]』

ヘネシーはGoogleの親会社Alphabet会長、パターソンはRISC-V財団副会長かつGoogleのExtinguished Engineerとして、オープンなアーキテクチャRISC-Vを解説すると共に、Googleのウェアハウススケールコンピュータ(WSC)を総合的実例として本書で初めて世界に公表した。

本書はこのテーマの世界最高峰のテキストだが、2人による最後のブック形式の書と目されている。

もっと少なく読む

本の長さ



472ページ

言語



日本語

出版社



星雲社

発売日



2019/9/25

ISBN-10



4434264001



<https://www.amazon.co.jp/%E3%82%B3%E3%83%B3%E3%83%94%E3%83%A5%E3%83%BC%E3%82%BF%E3%82%A2%E3%83%BC%E3%82%AD%E3%83%86%E3%82%AF%E3%83%81%E3%83%A3-%E7%AC%AC6%E7%89%88-%E5%AE%9A%E9%87%8F%E7%9A%84%E3%82%A2%E3%83%97%E3%83%AD%E3%83%BC%E3%83%81-%E3%82%B8%E3%83%A7%E3%83%B3-%E3%83%98%E3%83%8D%E3%82%B7%E3%83%BC/dp/4434264001>

今日の内容

- 1)プログラミングと開発プロセス
- 2)内部設計の前回残り
- 3)コーディング概要
- 4)プログラミング書法**

4) プログラミング書法と作法

- 良いプログラミングを行ううえでの基本的な規則やスタイルをプログラミング書法やプログラミング作法と呼び、主な目的は読みやすいプログラムを書く事である。

利点1: プログラムを作成した本人のため

- 実装中、バグに気付くかも
- 1年前の自分のコードは覚えてない

利点2: システムの共同開発者とのコミュニケーションを容易にし、開発後の保守やメンテナンスをしやすくする。

- プログラム書法の原則

① シンプルにわかりやすく記述

② 構造化プログラミング

順次、分岐、繰返しでプログラム作成する。
「抽象度の高い処理」と等価の「より詳細な処理」を作成

③ 特定のハードウェアやコンパイラ、ライブラリに依存しない

④ わかりやすいコメントを記述

例: • Javaの開発元が買収されて、使用ライセンスが変わっても大丈夫なように。
• ライブラリなどがメンテナンス(OS対応やセキュリティ対策)されなくなるかも

4) プログラムの表現

- プログラム作成時の一般的な注意事項として下記がある。プログラムを読む人にとってわかりやすくなること、間違いや誤解を生じないようにするのが主な目的である。

a) 変数の名前

① 混同しない名前を用いる

例: errFlagはTrueだとエラーが無いの？エラー有るの？

② 意味のある変数名を用いる

例: aとかbでなく、英単語使う

b) 定数および変数の設定

例: 定数使う場合は、マクロ(#define文)使う

4) プログラムの制御構造

- 制御構造(順次、条件分岐、繰り返し)の組合せで、どんなプログラムも記述できるとされているが、避けるのが望ましい構造もある。

1)条件分岐:なるべく上から下へと単純に読み進めていけるものがよく、途中で他の部分へジャンプしたりすることは避けるのが望ましい。

→gotoは基本的に使わない。

2)繰り返し構造:

a) モジュール論理構造の明確化

→ 順次、分岐、反復の区切りがわかりやすいのが望ましい。

b) プログラム構造の設計

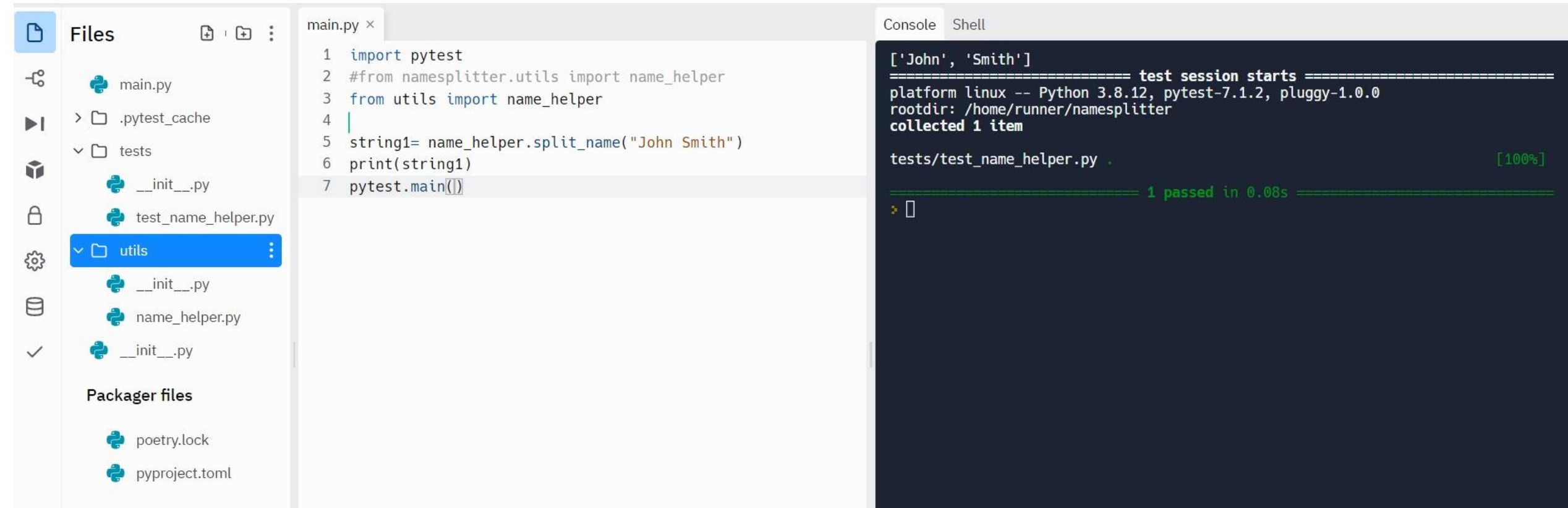
→ 複雑なプログラムを作る場合は、擬似コード(実際のプログラミング言語でない表記)で、全体像を把握する。

4) コメントの必要性

- ちょっと複雑なプログラムだと、1年前、半年前の自分のプログラムは、初めて見るような状態になる場合が多いです。自分のためにも、(今の自分にとっては当たり前的事も)細かくコメントするようにしましょう。
- **コメント例:**
 - モジュール(例:クラス、関数、プログラムファイル、構造体)の先頭には、モジュール概要と機能を記述
 - プログラムからでは直接読み取りにくい部分に記載
 - #変数の制約(例:フラグ数値の範囲と意味)、プログラムの制約
 - コメントは、必ずプログラムと一致させる。誤ったコメントは困る。。

(余裕あれば)課題5-3単体テスト(締切:6/2)

replit環境で、単体テストを行い、実行結果(テスト成功,テスト失敗)のスクリーンショットを提出ください。



```
main.py ×
1 import pytest
2 #from namesplitter.utils import name_helper
3 from utils import name_helper
4
5 string1= name_helper.split_name("John Smith")
6 print(string1)
7 pytest.main()
```

```
['John', 'Smith']
===== test session starts =====
platform linux -- Python 3.8.12, pytest-7.1.2, pluggy-1.0.0
rootdir: /home/runner/namesplitter
collected 1 item

tests/test_name_helper.py . [100%]

===== 1 passed in 0.08s =====
```

(余裕あれば)課題5-3単体テスト(締切:6/2)

Google

replit

すべて ニュース 画像 動画 ショッピング

約 1,240,000 件 (0.78 秒)

https://replit.com ▼

Replit: The collaborative browser based IDE

Replit is a simple yet powerful online IDE, Editor, Compiler, Interpreter, compile, run, and host in 50+ programming languages.
このページに 4 回アクセスしています。前回のアクセス: 22/04/1

[Log In](#)

Replit is a simple yet powerful online IDE, Editor, Compiler ...



replit Features ▼ Blog Pricing Jam Teams Pro Teams for Education Careers

[Log in](#) [Sign up](#)

Code, create, and learn together

Use our free, collaborative, in-browser IDE to code in 50+ languages — without spending a second on setup.

[<> Start coding](#)

(余裕あれば)課題5-3単体テスト(締切:6/2)

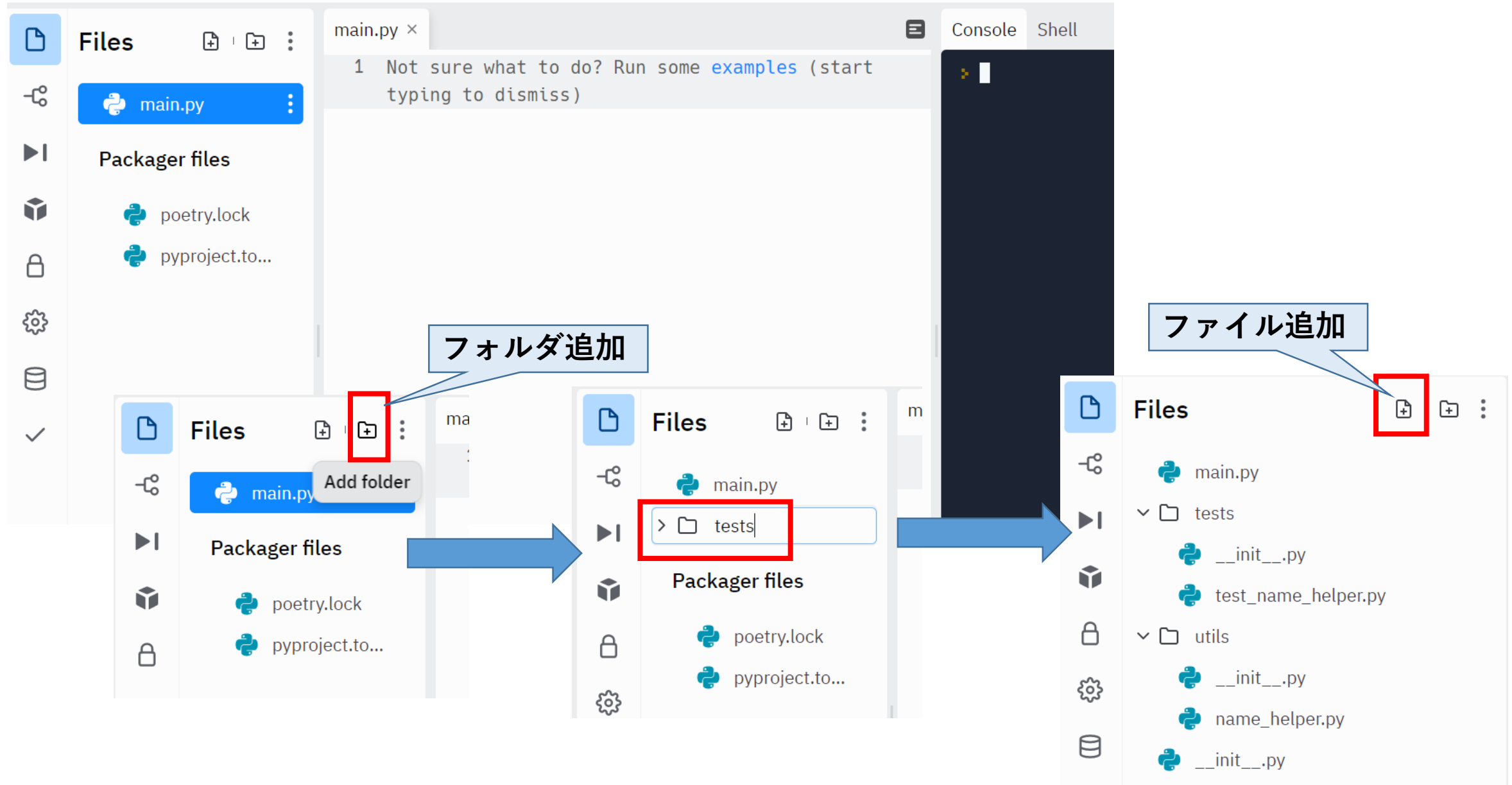
The image shows a sequence of three screenshots from the Replit website, illustrating the process of creating a new REPL (Python environment).

Left Screenshot: The user is logged in as @tsuchidat. The **+ Create** button is highlighted with a red box. A blue arrow points down to the **Python repl** option under the **Official Languages** section, which is also highlighted with a red box.

Middle Screenshot: The **Create a repl** form is shown. The **Template** is set to **Python**. The **Title** field is labeled "Name your repl". The **Privacy** is set to **Public**. A blue arrow points from the **Python repl** option in the left screenshot to the **Python** template in this form.

Right Screenshot: The **Create a repl** form is shown with the **Title** field set to **namesplit**, which is highlighted with a red box. A blue arrow points from the **namesplit** title to the **+ Create Repl** button at the bottom, which is also highlighted with a red box.

(余裕あれば)課題5-3単体テスト(締切:6/2)



(余裕あれば)課題5-3単体テスト(締切:6/2)

The image shows a development environment with a file explorer on the left and three code snippets on the right, connected by blue arrows.

File Explorer:

- `main.py` (highlighted with a red box)
- `tests` folder
 - `__init__.py`
 - `test_name_helper.py` (highlighted with a red box)
- `utils` folder
 - `__init__.py`
 - `name_helper.py` (highlighted with a red box)
 - `__init__.py`

Code Snippets:

`main.py`:

```
1 import pytest
2 from utils import name_helper
3
4 string1= name_helper.split_name("John Smith")
5 print(string1)
6 pytest.main()
```



`tests/test_name_helper.py`:


```
1 from namesplit.utils import name_helper
2
3 def test_two_names():
4     assert name_helper.split_name("John Smith") == ["John", "Smith"]
```


`utils/name_helper.py`:

```
1 def split_name(name):
2     first_name, last_name = name.split()
3     return [first_name, last_name]
```

Callout: 返り値が["John", "Smith"]になることを確認

it  



Invite 

main.py ×

Console Shell

```
1 import pytest
2 from utils import name_helper
3
4 string1= name_helper.split_name("John Smith")
5 print(string1)
6 pytest.main()
```

['John', 'Smith']

===== test session starts =====




platform linux -- Python 3.8.12, pytest-7.1.2, pluggy-1.0.0
rootdir: /home/runner/namesplit
collected 1 item


tests/test_name_helper.py .
[100%]

===== 1 passed in 0.09s =====

>

テスト成功

plit   ▶ Run 👤 Invite 

utils/name_helper.py ×  Console Shell

```
1 ▼ def split_name(name):
2     #     first_name, last_name = name.split()
3     #     return [first_name, last_name]
4     ||| return name
```

tests/test_name_helper.py ×

collected 1 item

tests/test_name_helper.py F
[100%]

===== FAILURES =====

test_two_names

```
def test_two_names():
>     assert name_helper.split_name("John Smith") == ["John", "Smith"]
E       AssertionError: assert 'John Smith' == ['John', 'Smith']
E         + where 'John Smith' = <function split_name at 0x7fa8196ebdc0>('John Smith')
E         + where <function split_name at 0x7fa8196ebdc0> = name_helper.split_name

tests/test_name_helper.py:4: AssertionError
===== short test summary info =====
FAILED tests/test_name_helper.py::test_two_names - AssertionError: assert 'Jo...
```

1 failed in 0.21s

🗨

テスト失敗

提出:テスト成功とテスト失敗のスクリーンショットを提出ください

