プログラミング言語設計論

2014年度

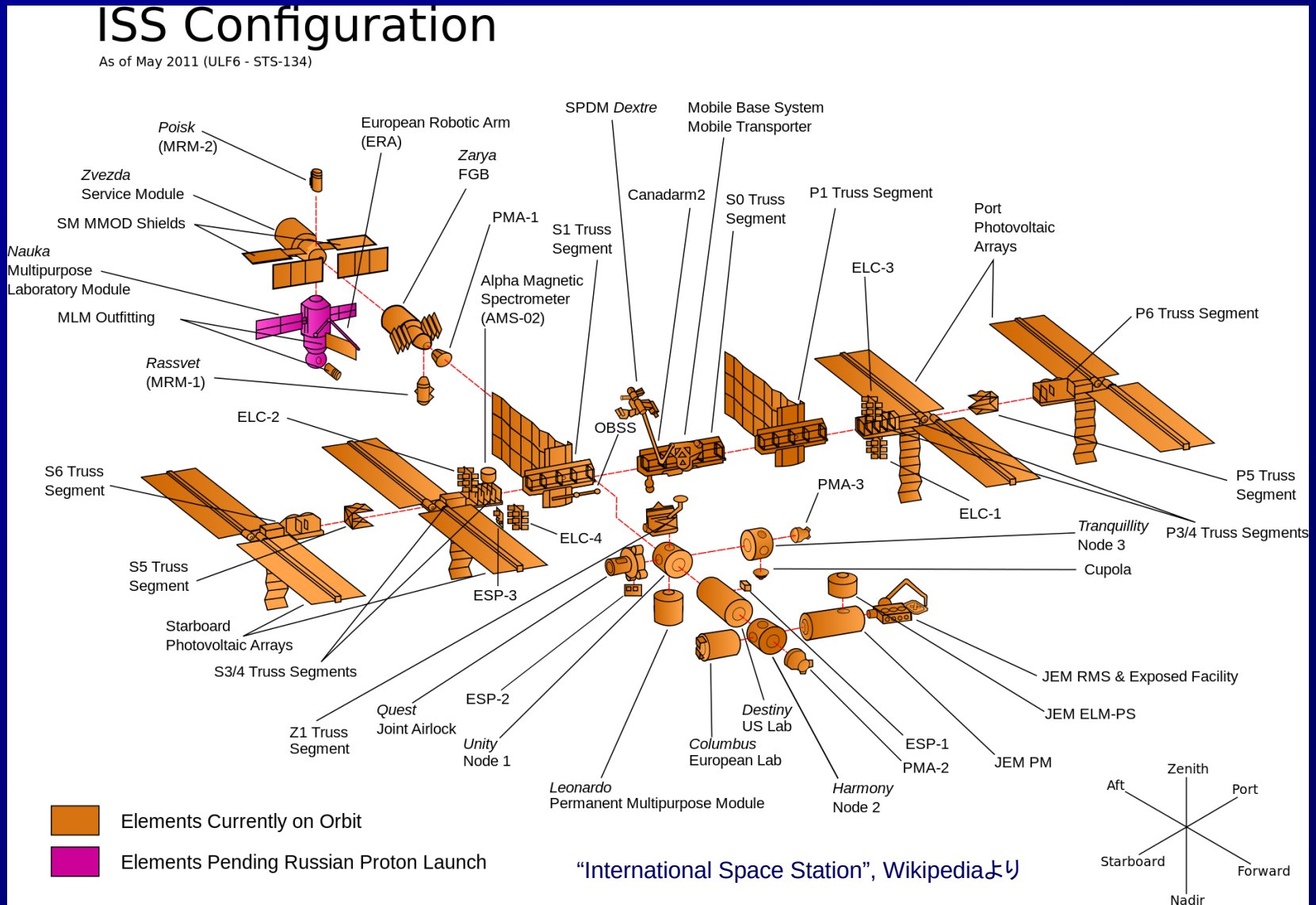
第1週: モジュール化

担当: 増原英彦

モジュール

- 定義 (一般的な「モジュール」):
  - 大きなシステムの部分を構成するもの
  - 他のモジュールとの連携方法が規格化されている
- 特徴:
  - 開発・交換・保守の単位
  - 「使う人」と「作る人」がいる
  - 入れ子になることもある
  - 多くの場合は物理的な単位になる

モジュールの例:
国際宇宙ステーション



“International Space Station”, Wikipediaより

例題: Key Word in Context (KWIC)システム [Par72]

出力

hile allowing the shortening of
s. An alternative approach be
discusses modularization as a
echanism this paper discusses
is time The effectiveness of a
or more subroutines, will that a
problem is presented and into
at a module consists of one or
e approach be less efficient in

:

its development system while allowin
less efficient in most cases. An alter
mechanism this paper discusses mo
modularization as a mechanism this
modularization is time The effectiven
module consists of one or more subr
modules. A system design problem
more subroutines, will that a module
most cases. An alternative approach

:

例題: Key Word in Context
(KWIC)システム

出力

	:
hile allowing the shortening of	its development system while allowin
s. An alternative approach be	less efficient in most cases. An alter
discusses modularization as a	mechanism this paper discusses mo
echanism this paper discusses	modularization as a mechanism this
is time The effectiveness of a	modularization is time The effiecten
or more subroutines, will that a	module consists of one or more subr
problem is presented and into	modules. A system design problem
at a module consists of one or	more subroutines, will that a module
e approach be less efficient in	most cases. An alternative approach
	:

5

アルゴリズム

- ファイルを行単語で読み込む
- 各行について、0～w単語だけ回転させた文字列の配列を作る。全ての行についてそれを行い、結合した配列を作る
- その配列を整列する
- その配列を順に出力する

本日の問題(1/3)

- 配布した紙に氏名・学籍番号を記入
- 問1: KWICシステムを、全体を5個程度のモジュールで構成する設計を考えよ。各モジュールの役割と、モジュールが用意する機能(関数・メソッド)の役割を書け

アルゴリズム

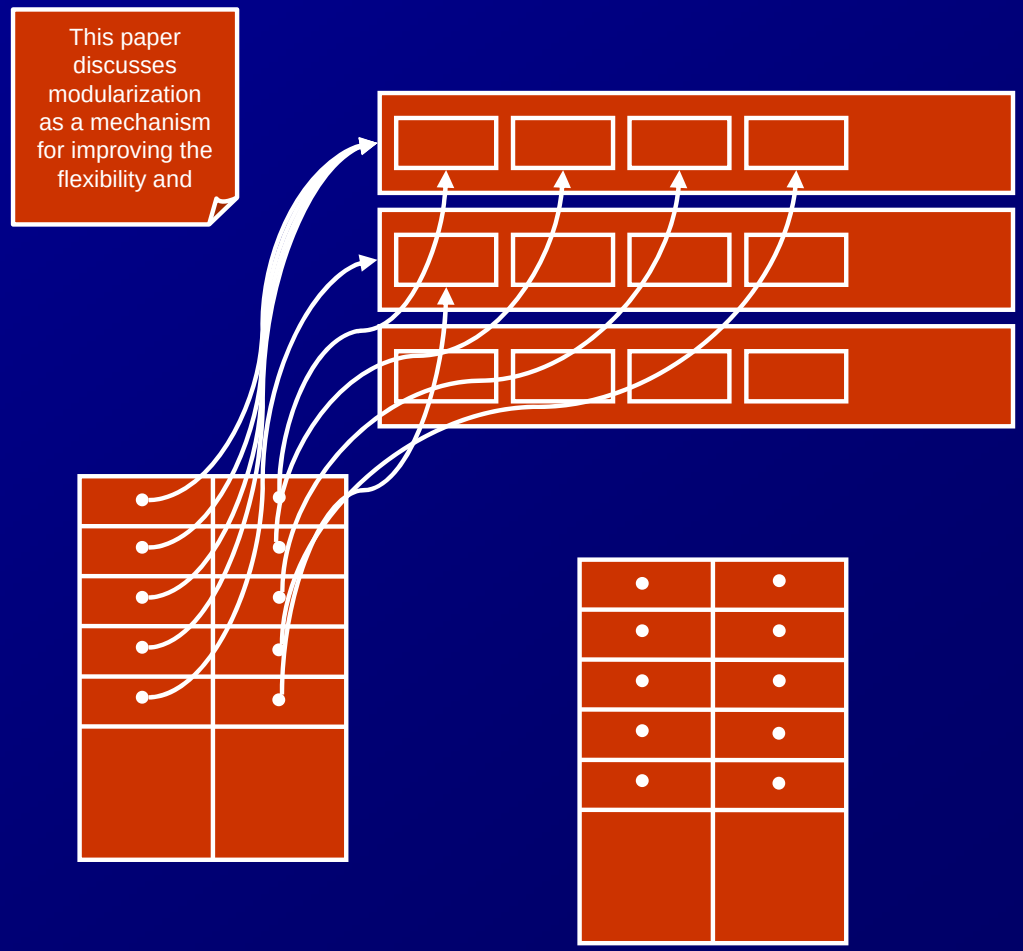
- ファイルを行単語で読み込む
- 各行について、0～w単語だけ回転させた文字列の配列を作る。全ての行についてそれを行い、結合した配列を作る
- その配列を整列する
- その配列を順に出力する

本日の問題(2-3/3)

- 問2: ソフトウェアの設計を「どちらがより良くモジュール化されているか」という点で比較する方法を考えて書け。
- 問3: オープンソースソフトウェアと商用ソフトウェアでは、どちらが「より良くモジュール化されている」だろうか? 理由とともに書け。ただし同程度に大規模なものを比較したとする。

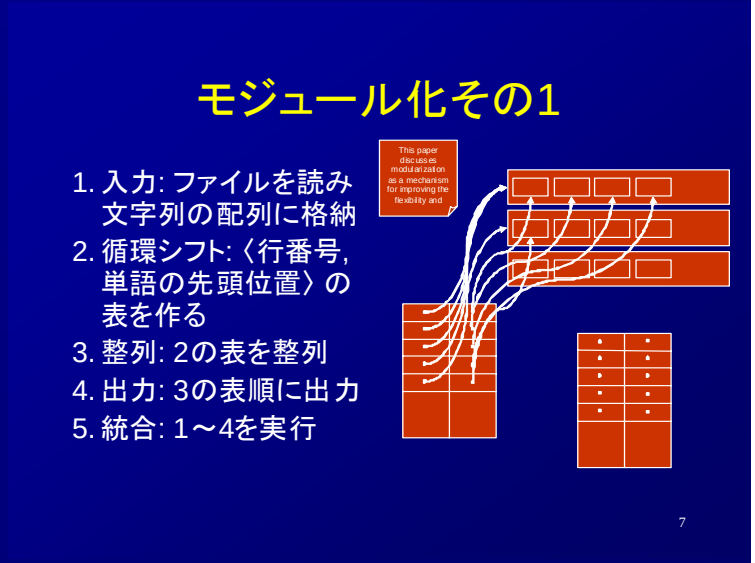
モジュール化その1

1. 入力: ファイルを読み文字列の配列に格納
2. 循環シフト: 〈行番号, 単語の先頭位置〉の表を作る
3. 整列: 2の表を整列
4. 出力: 3の表順に出力
5. 統合: 1～4を実行



モジュール化その1

- データ構造:
  - 行配列: String[]
  - シフト表: int[][]
- インタフェース
  - 入力: String[] input(String fileName)
  - 循環シフト: int[][] shift(String[] lines)
  - 整列: int[][] sort(String[] lines, int[][] shift)
  - 出力: void output(String[] lines, int[][] shift)
  - 統合: void main()



モジュール化その2

1. 行記憶
   - CHAR(r,w,c) r行目のw単語目のc文字目
   - SETCHAR(r,w,c,d), WORDS(r), DELINE, DELWRD
2. 入力: ファイルから行を読み、行記憶に格納
3. 循環シフト:
   - CSCHAR(l,w,c) l番目のシフトのw単語目のc文字目
   - CSSETUP
4. 整列
   - ITH(i) 整列後i番目に来るシフト番号
   - ALPH
5. 出力
6. 統合管理

モジュール化その2

1. 行記憶
   - char getChar(int row, int word, int offset)
   - void setChar(int row, int word, int offset, char c)
   - int numWords(int row)
2. 入力: input(String fileName)
3. 循環シフト:
   - void initialize()
   - char getChar(int shift, int word, int offset)
   - int numWords(int shift)
4. 整列
   - void doSort()
   - int getShift(int i)
5. 出力: void output()
6. 統合管理: void main()

どちらがよいのか?

モジュール化その1

- データ構造:
  - 行配列: String[]
  - シフト表: int[][]
- インタフェース
  - 入力: String[] input(String fileName)
  - 循環シフト: int[][] shift(String[] lines)
  - 整列: int[][] sort(String[] lines, int[][] shift)
  - 出力: void output(String[] lines, int[][] shift)
  - 統合: void main()

モジュール化その2

1. 行記憶
   - char getChar(int row, int word, int offset)
   - void setChar(int row, int word, int offset, char c)
   - int numWords(int row)
2. 入力: input(String fileName)
3. 循環シフト:
   - void initialize()
   - char getChar(int shift, int word, int offset)
   - int numWords(int shift)
4. 整列
   - void doSort()
   - int getShift(int i)
5. 出力: void output()
6. 統合管理: void main()

Parnasモジュール化原則 [Par72]

... begins with a list of difficult design decisions or design decisions which are likely to change. Each module is then designed to hide such a decision from the others.

起こり得る変更

- 全ての行をメモリに読み込む←→読み込まない
- 文字列の表現形式: String ←→バイト列のまま
- シフトの表現:文字列の複製を作る←→ポインタを使う
- 整列の範囲: 全てを一度に←→on demandで

モジュール化そ

1. 入力: ファイルを読み文字列の配列に格納
2. 循環シフト: 〈行番号, 単語の先頭位置〉の表を作る
3. 整列: 2の表を整列
4. 出力: 3の表順に出力
5. 統合: 1〜4を実行

その1

- データ構造:
  - 行配列: String[]
  - シフト表: int[][]
- インタフェース
  - 入力: String[] input(String fileName)
  - 循環シフト: int[][] shift(String[] lines)
  - 整列: int[][] sort(String[] lines, int[][] shift)
  - 出力: void output(String[] lines, int[][] shift)
  - 統合: void main()

起こり得る変更

- 全ての行をメモリに読み込む←→読み込まない
- 文字列の表現形式: String ←→バイト列のまま
- シフトの表現: 文字列の複製を作る←→ポインタを使う
- 整列の範囲: 全てを一度に←→on demandで

その2

1. 行記憶
   - char getChar(int row, int word, int offset)
   - void setChar(int row, int word, int offset, char c)
   - int numWords(int row)
2. 入力: input(String fileName)
3. 循環シフト:
   - void initialize()
   - char getChar(int shift, int word, int offset)
   - int numWords(int shift)
4. 整列
   - void doSort()
   - int getShift(int i)
5. 出力: void output()
6. 統合管理: void main()

モジュール化の程度を測る

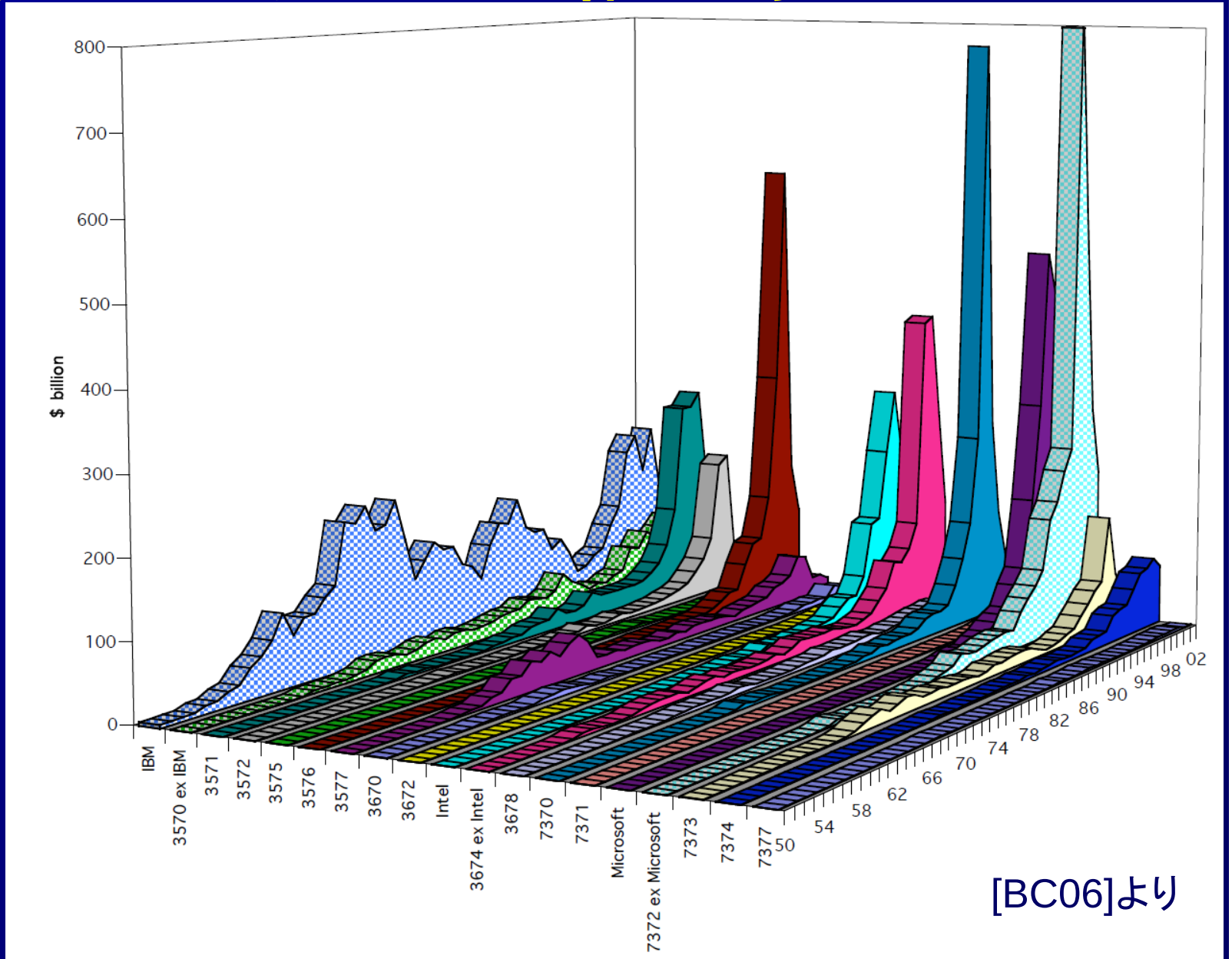
- AとBどちらがよりモジュラーかをどうやって測ればよいのか?

他分野での手法: 経営学

- 産業構造のモジュール化の程度 [BC06]
- 自動車産業と自転車産業のモジュール化 [藤本04]

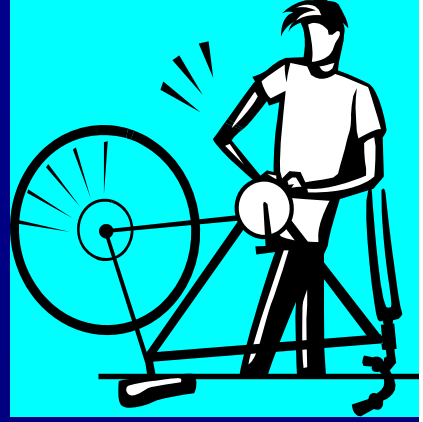
情報通信産業の構造変化

モジュール化が進む
↓
経済規模が大発展

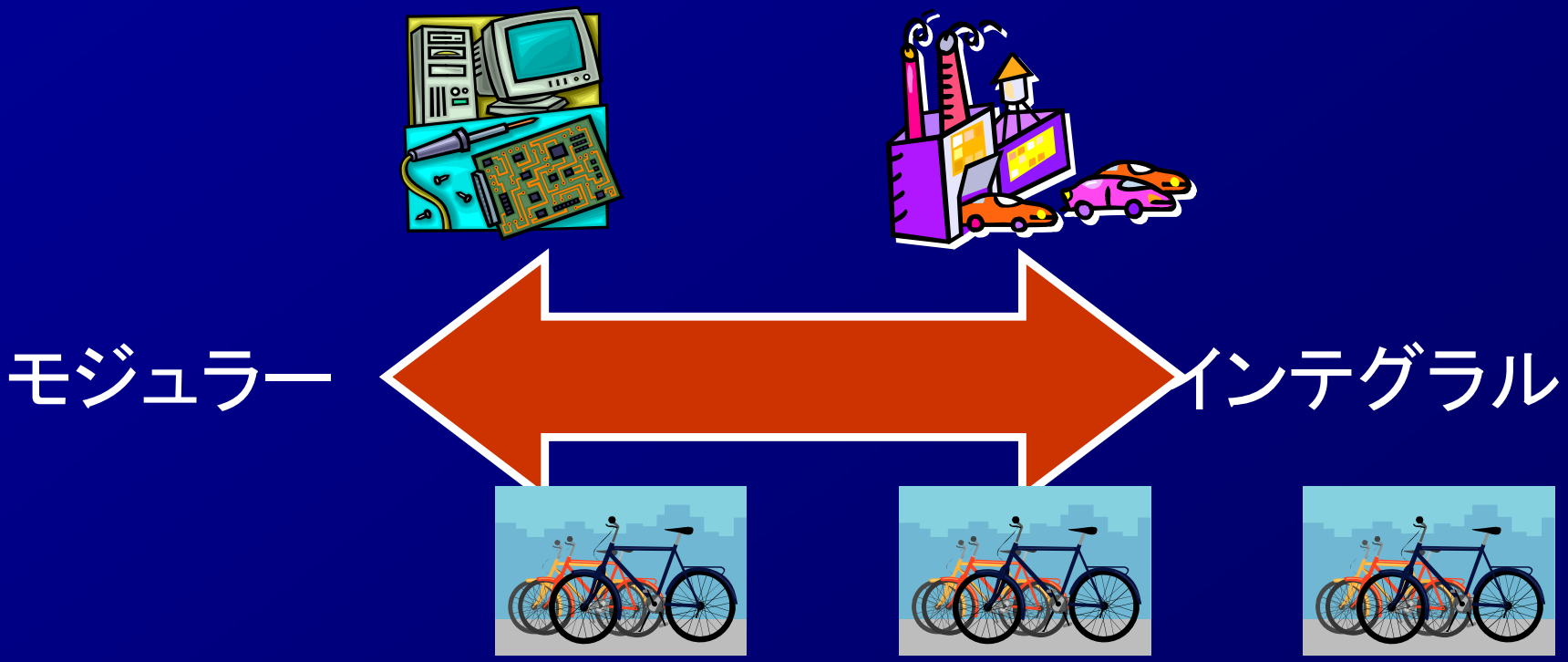


[BC06]より

産業構造とモジュール化



- 自動車と自転車、どっちがモジュラー?



モジュール性の測定方法

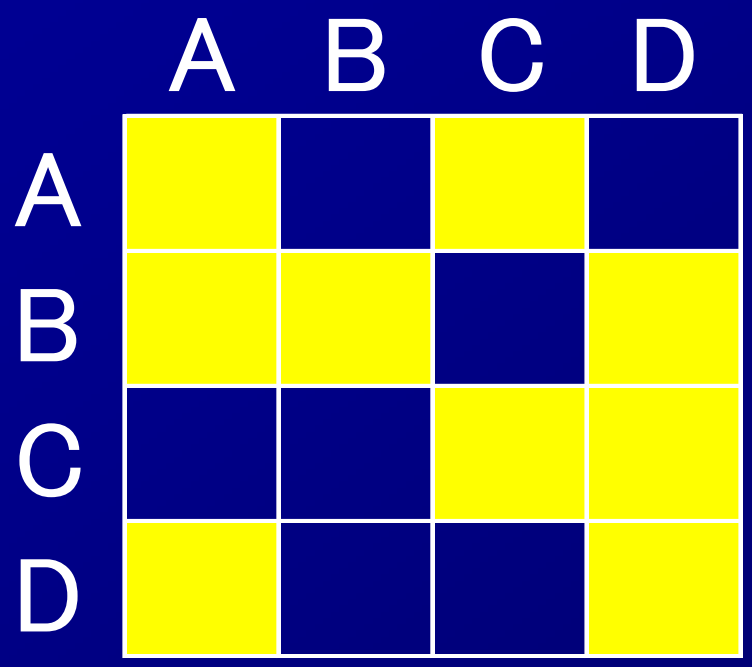
- Parnas原則:
  困難な/変わりやすい決定を隠せ
- 指標の1つ: 変化時に受ける影響の大きさ
  - 何の変化: あるモジュールの化ンタフェース
  - 影響の大きさ: 修正しなければいけない
    モジュールの個数

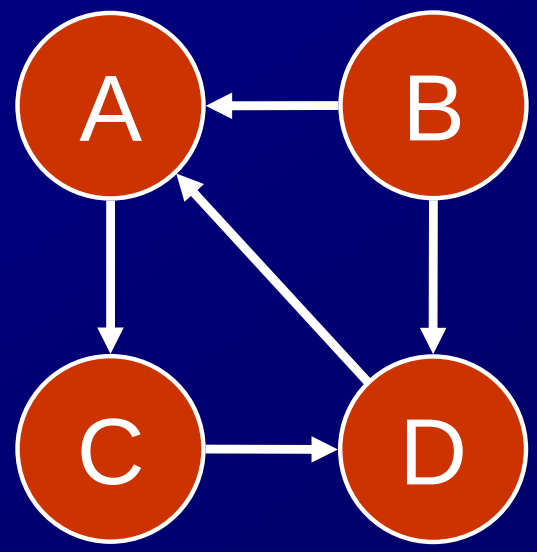
モジュール性の測定方法
設計構造行列と変更影響分析

- 設計構造行列 (design structure matrix)
- 変更影響分析 (change impact analysis)

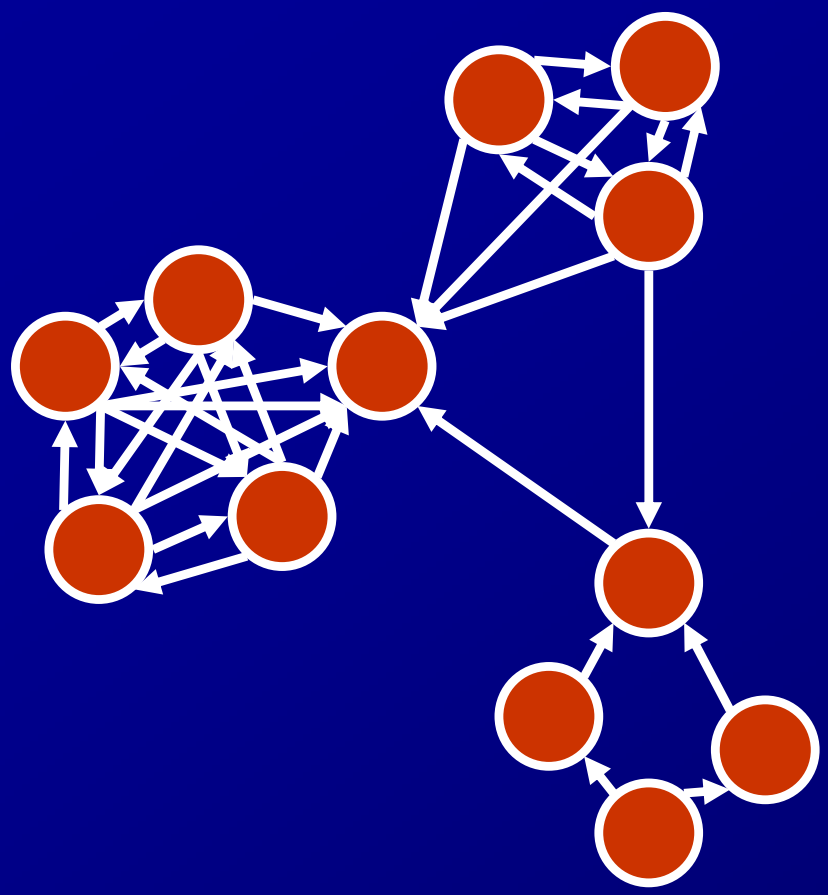
設計構造行列 (グラフ) [BC06]

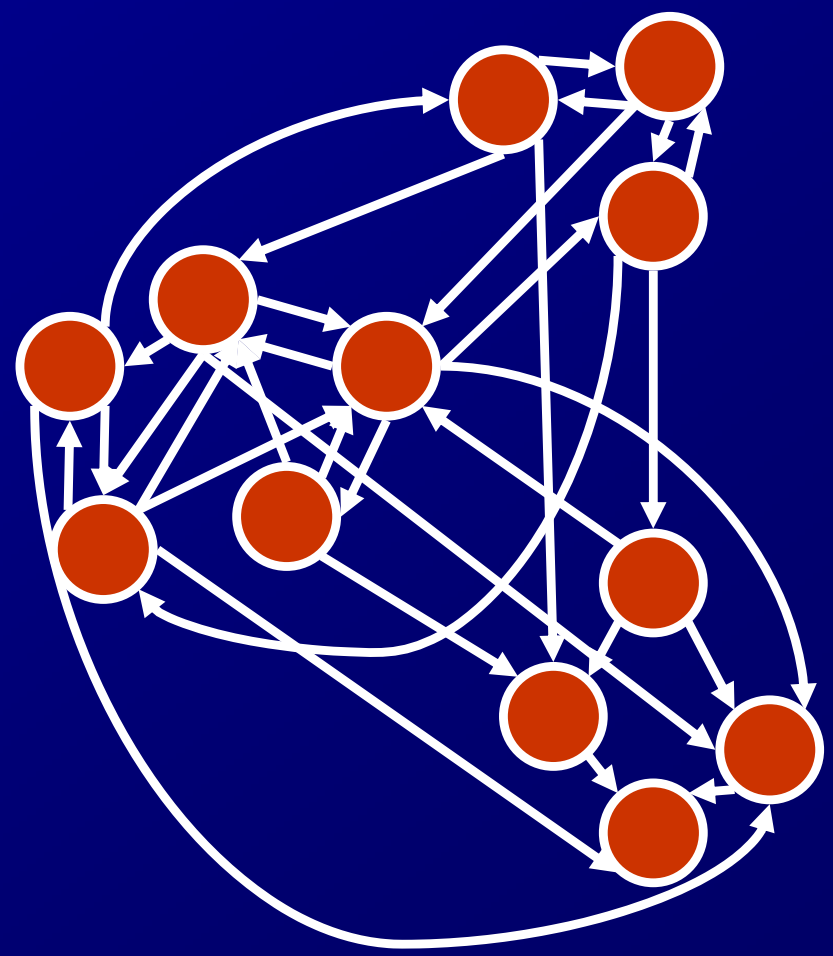
■i行j列→iがjに依存





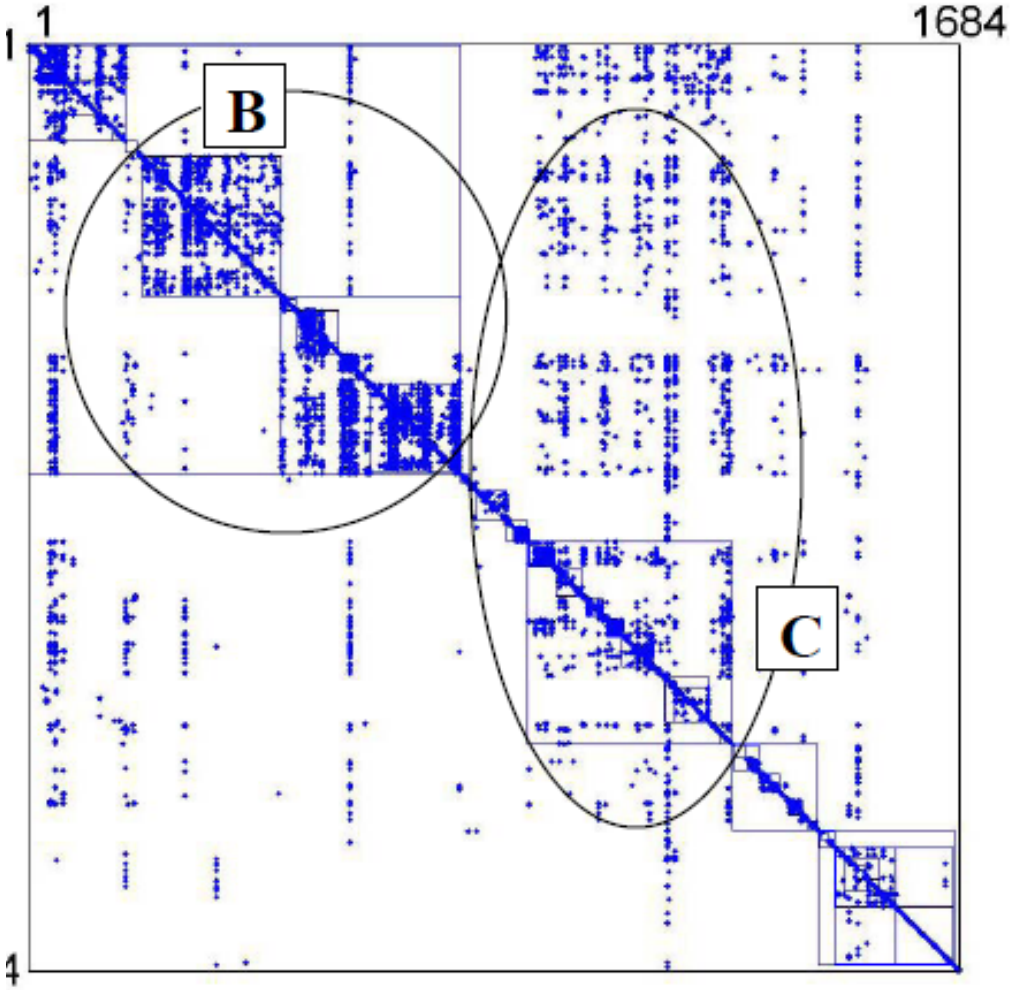
設計構造で見たモジュール化の良し悪し



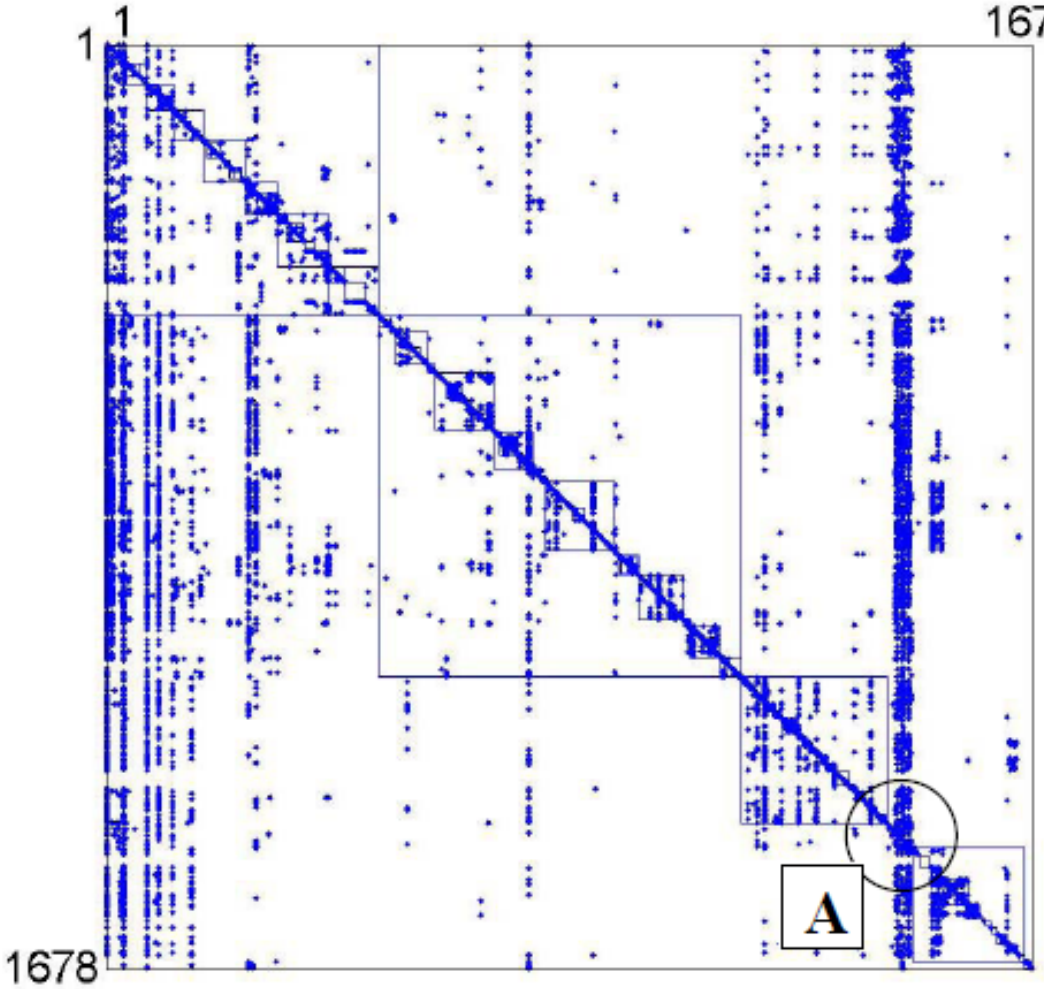


設計構造行列による分析 [MRC06]

mozilla.19980408

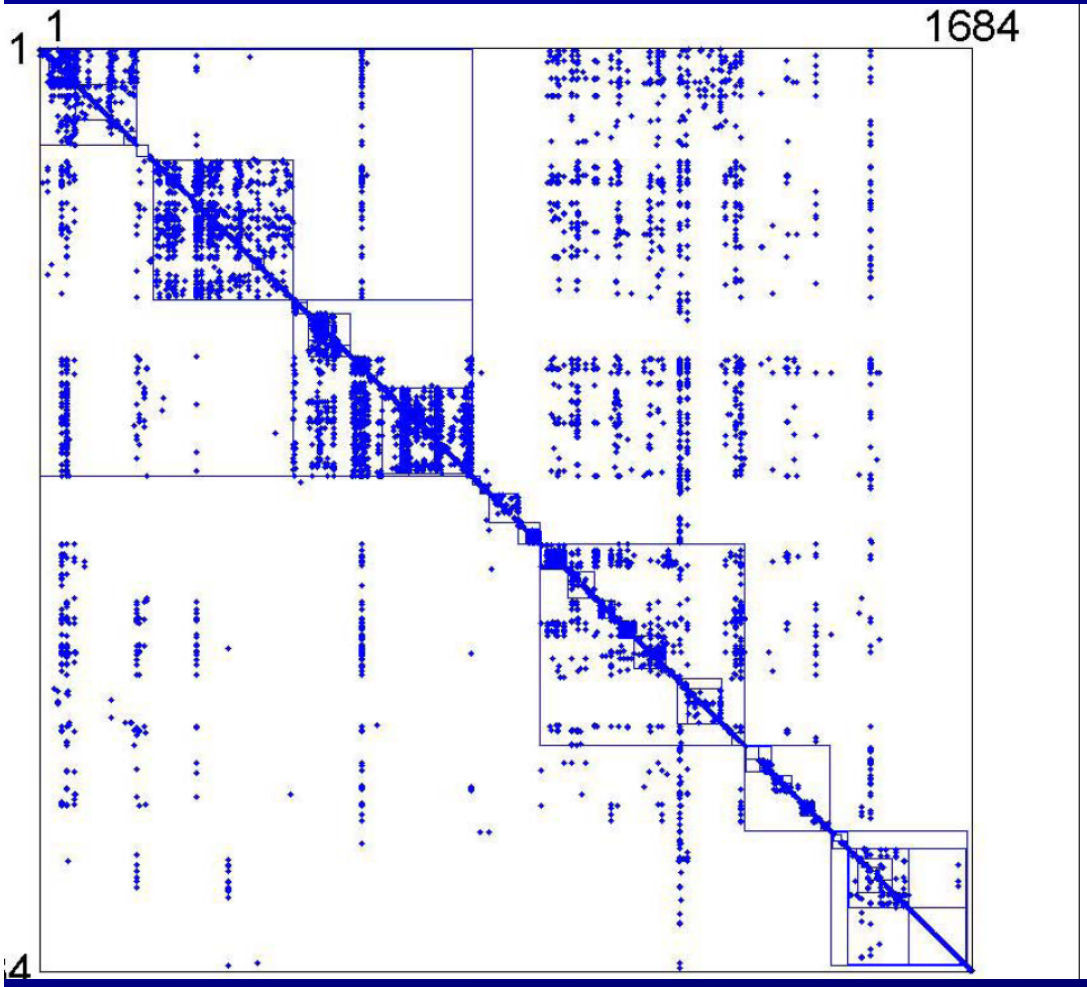


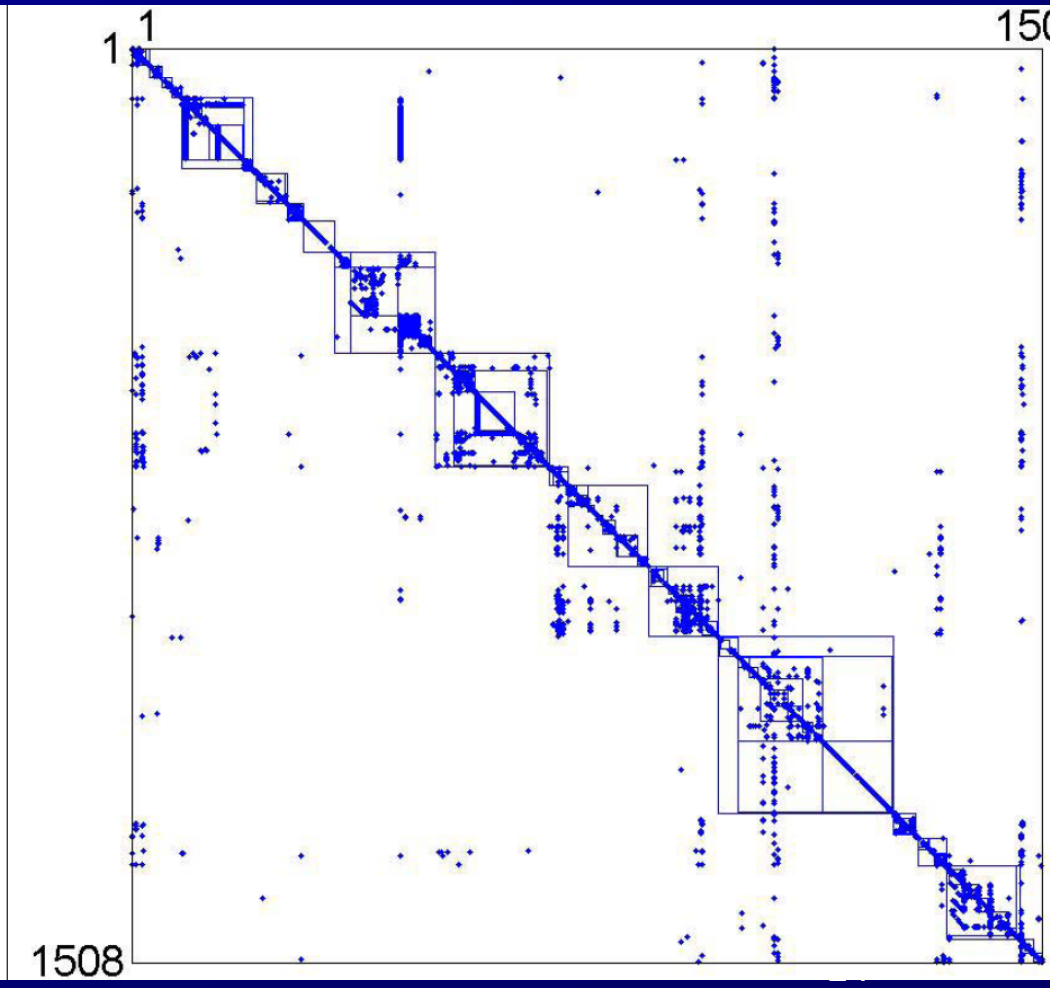
linux.2.1.105



設計構造行列の時間変化

- mozilla.19980408
- mozilla.19981211



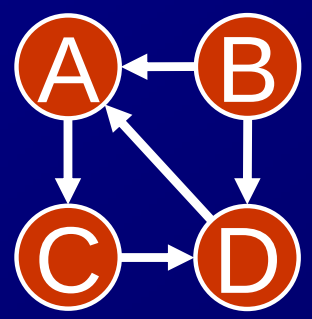


モジュール化の程度の数値化
変更影響分析

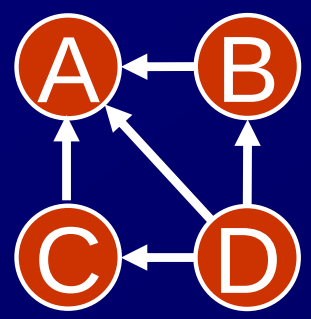
- いろいろある
- 影響伝搬費用 (propagation cost)
  - M: モジュールの集合
  - d(m): mが直接間接に依存するモジュールの個数

$$\frac{\sum_{m \in M} |d(m)|}{|M|^2}$$

変更の影響が推移的と仮定



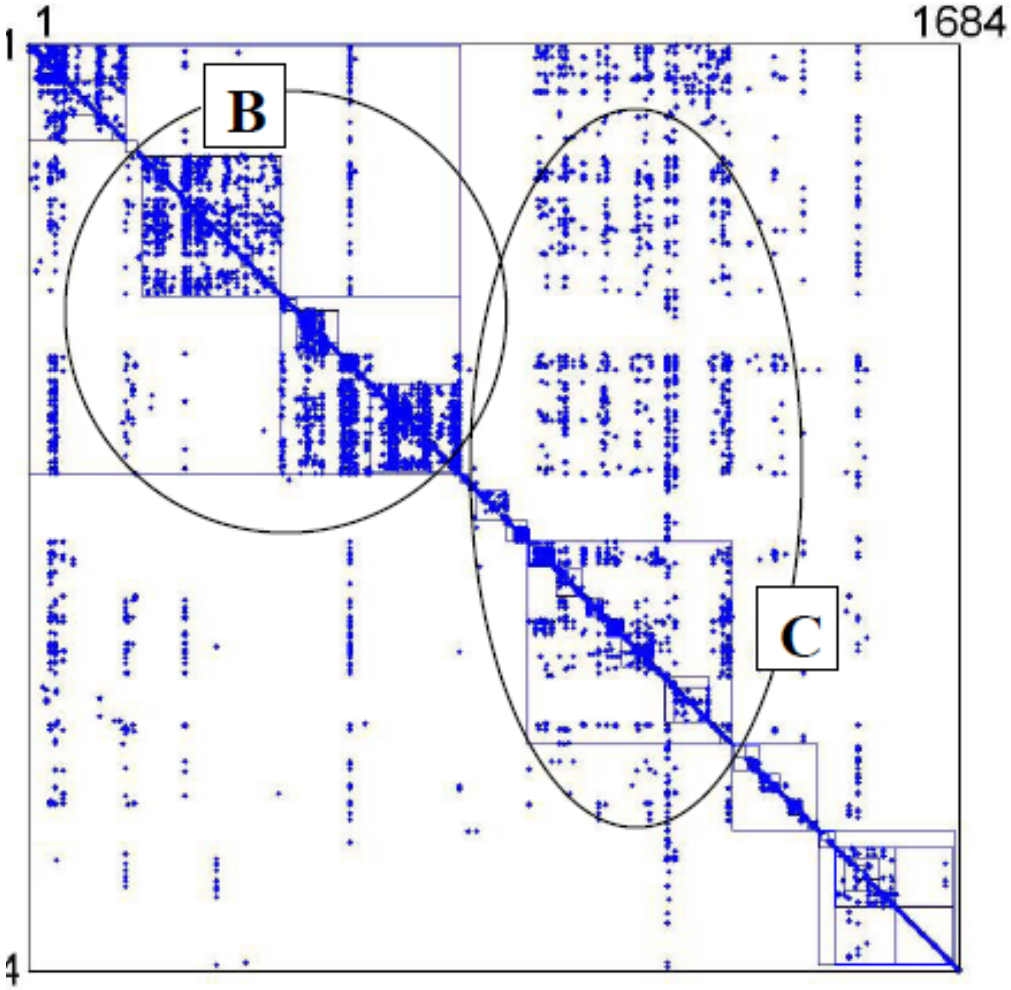
13/16=81%



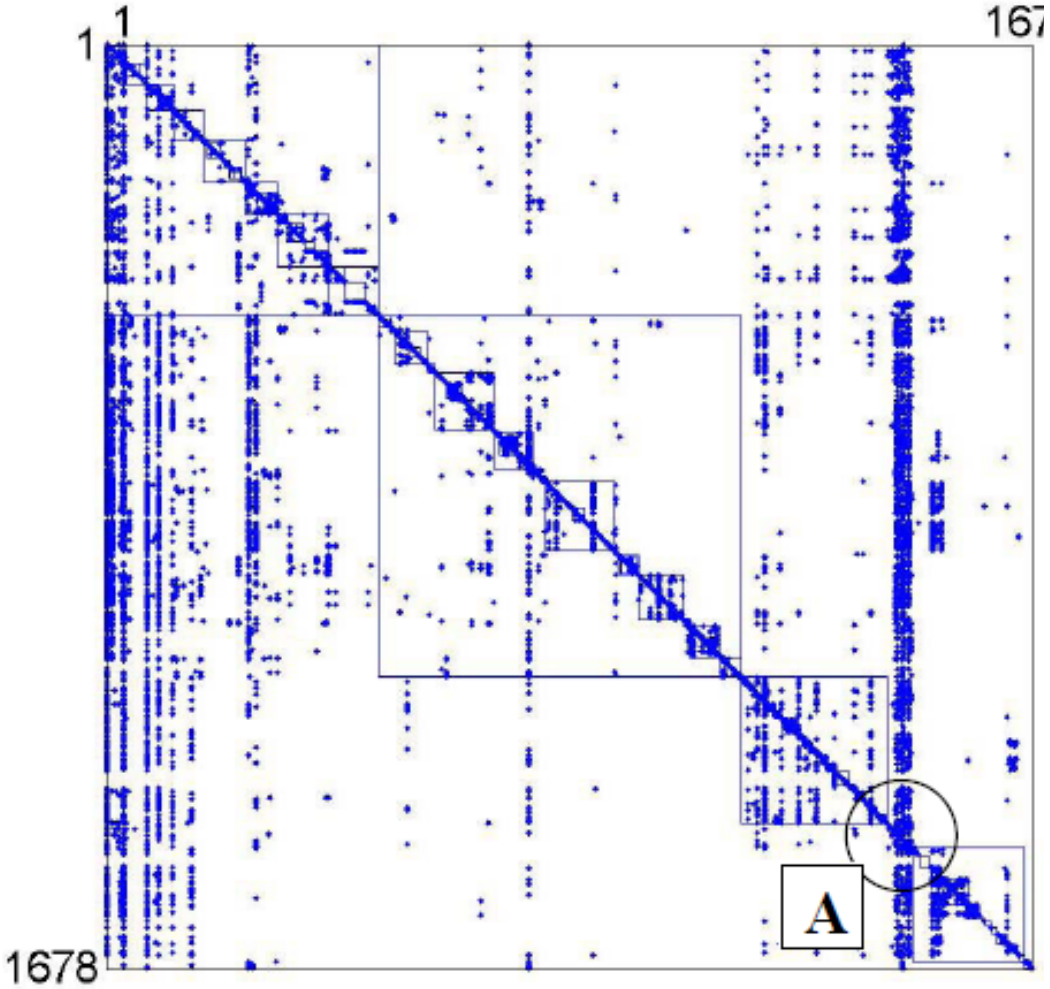
5/16=31%

設計構造行列による分析

mozilla.19980408 17.35%

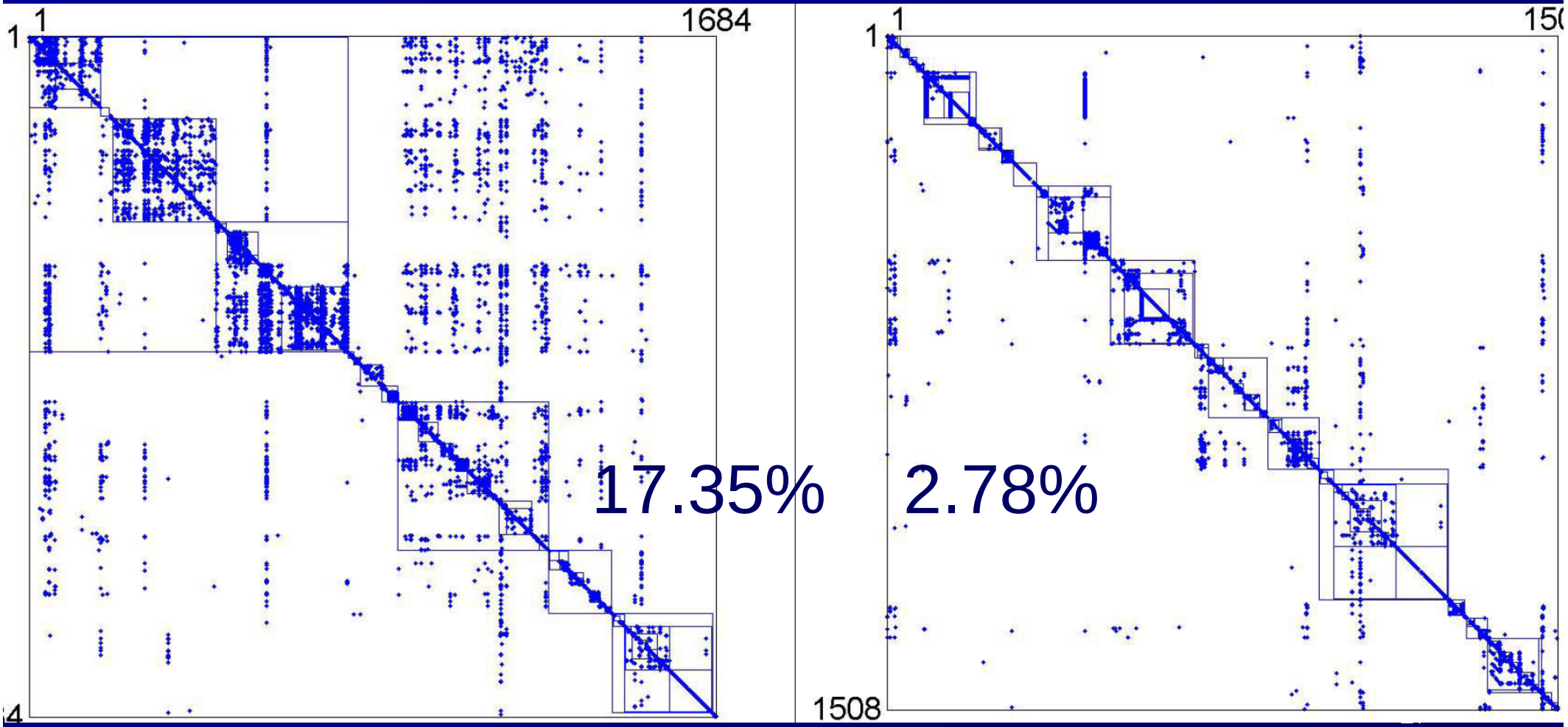


5.82% linux.2.1.105



設計構造行列の時間変化

- mozilla.19980408
- mozilla.19981211



観察結果

同規模のソフトウェアを比較

- ➢オープンソースプログラム
- ➢商用プログラム

■前者の方がモジュール化が進んでいる

■ソフトウェアを再構築すると
モジュール化が進む

Open question: その理由は?

参考文献

[BC06] Carliss Y. Baldwin and Kim B. Clark, “Modularity in the Design of Complex Engineering Systems,” in *Complex Engineered Systems: Science Meets Technology*, Springer, 2006.

[MRC06] Alan MacCormack, John Rusnak and Carliss Y. Baldwin, “Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code,” in *Management Science* 52(7), 2006.

[Par72] D. L. Parnas, “On the Criteria to be Used in Decomposing Systems into Modules”, in *Communications of the ACM* 15(12)1053-1058, 1972.

[藤本04] 藤本隆宏, 日本のもの造り哲学, 日本経済新聞社, 2004