大学におけるプロセス改善教育のあり方について
– PSP 法実践の経験をもとに –

海谷 治彦　　小島 彰　　海尻 賢二

信州大学 工学部 情報工学科

{kaiya,cakira,kaijiri}@cs.shinshu-u.ac.jp

http://www.cs.shinshu-u.ac.jp/~kaiya/

あらまし

大学でのソフトウェア教育において，開発プロセスの定義，測定，改善をとりあがることは望ましいと思われる．しかし，大学において組織規模の実践を行い，その実践を動機付けることは困難である．本稿では，大学院における PSP 法の演習の報告を行い，大学においてプロセス改善の実習と，改善活動への動機付けをどのようにして行うべきかについて述べる．

1　はじめに

大学でのソフトウェア教育において，開発プロセスの定義，測定，改善がとりあげられることはほとんどない．実際，国内，情報処理・情報工学系学科のカリキュラム作成に大きな影響を与えている「大学の理工系学部情報系学科のためのコンピュータサイエンス教育カリキュラム J97」[1] でも，全く取り上げられていない．しかし，プロセスに関する学習を大学で行う必要がないわけではないと思われる．例えば，SWEBOK の調査 [2] でも，Project Management が重要な知識とみなされているにもかかわらず，その知識取得が大学教育であまり行われていないことが示されている．また，次世代カリキュラムの雛型となる ACM，IEEE/CS による Computing Curricula 2001[3] では，ソフトウェア工学の第 1 項目としてプロセスとメトリクスがとりあげられている．

ソフトウェア開発プロセスとその改善に関する教育を大学で十分に行うことができない第一の理由として，学生自身の言語や設計法など開発対象に対する技能や知識が十分でないことが挙げられる．しかし，もし開発技法に関する知識を既に習得済みであったとしても，プロセス改善に関する学習を行うのは困難だと思われる．その理由は，従来のプロセス改善に関する手法は，組織やチームなど，比較的大規模な集団に対する手法がほとんどだからである．大学の授業等の中では，そのような集団を組織するのは大変困難である．そこで，大学でプロセス改善の重要性の理解や，体験学習を行うためには，比較的，小規模，もしくは個人の開発活動に適用可能なプロセス技法が必要となる．

Watts Humphrey が提唱した PSP 法 [4, 5, 6, 7] では，個人レベルのプロセス定義，測定，見積もりの手法が提案されており，それを体感するための演習問題も添付されている．PSP が提唱された背景として，CMM などの組織に対するプロセス改善を実際に進めるためには，個々人の能力改善が必須であるという考え方がある．本手法はいくつかの大学での教育に利用されており，その報告もされている [8], [9]．しかし，これらの報告や，実際の PSP の遂行体験を鑑みるまでもなく，大学での半年程度の開発演習で，実際にプロセス改善が成されるはずがない．よって，改善成果以外の成果物や動機付けがなければ，本演習を継続することは困難である．また，本演習の遂行はかなりの労力を消費するため，現実的には実業務の合間に行うことは困難であると思われる．

本稿では，信州大学 工学部 情報工学系 大学院 (修士課程) における PSP 法の演習 [10] を通して，大学でのプロセス改善の教育では何をどのように行うべきかについて考察する．続く 2 節では PSP の概要について説明する．3 節では授業内容と受講者のプロセスデータを示す．そして，4 節で授業を行った感触や授業後にとったアンケートなどをもとに，大学におけるプロセス改善教育をどのように行うべきかについて述べる．

2　PSP の概要

PSP は，CMM のキープロセスエリアの内，特に個人能力に関係の深い部分の改善を促進するためのプロセス改善法である．PSP は，CMM と同様に成熟度のレベルを持っており，それは以下のようなものである．

- PSP0: ベースラインパーソナルプロセス
- PSP1: パーソナル計画立案プロセス
- PSP2: パーソナル品質管理
- PSP3: 循環的パーソナルプロセス

実際には PSP3 以外のステップは 2 つのサブステップに分割されている．

具体的な PSP の習得は，教科書 [5, 6] を用いて 10 個のプログラム開発演習を通して，行うことができる．表 1 に標準的な A コースの問題 10 個を示す．PSP では，開発時の時間，LOC，欠陥数などの測定値を用いて，開発規模，時間の見積もりを行うことが大きな比率を占めている．そのため，問題自体も，統計的な知識を学習す

表 1: PSP で利用する技法と演習問題の概要

PSP	測定技法	開発技法	問題	
0	時間・欠陥記録	ウォーターフォール開発	1A	期待値と標準偏差
0.1	LOC記録	コード標準	2A	LOCカウンタ
			3A	関数毎LOCカウンタ
1	規模・時間見積り (Probe)	**事前設計**	4A	線形回帰パラメータ
1.1	スケジュール見積もり		5A	シンプソン積分・正規分布
	欠陥見積もり, 見積もり予測範囲		6A	予測区間・t分布積分
2		**レビュー**	7A	相関とその有意性
2.1		設計図	8A	ソート
			9A	正規分布の検定
3		スパイラルな開発	10A	重回帰分析

るためのものが多い．

PSP のそれぞれのレベル (PSP0 から PSP3 まで) では，プロセス遂行のための手続きと，プロセス記録・分析のための表現形式が厳格に規定されている．それぞれにプロセススクリプトとフォームと教科書の中では呼ばれている．図 1 にその例を示す．これらのスクリプト遂行とフォーム記述を支援する公式のツールは存在しない．

PSP の演習過程では，表 1 に示すようなプロセス測定技法とプログラム開発技法を適用が義務付けられている．例えば，PSP0 では開発時間と欠陥の追跡 (作りこみと除去の個数と工程の記録) を行わなければならない．また，PSP2 では自分用に特化したレビューリストに基づき，設計レビュー，コードレビュー行うことが義務となっている．

PSP で最も特徴的な部分は，Probe 法と呼ばれる規模・時間見積り方法である．この見積もり法の考え方は非常に単純で，個人が過去に行ってきた開発記録をもとに，これから開発するであろうクラス (関数) がどの程度の行数となるかを予測するものである．クラス (関数) は 6 つのカテゴリ (計算，データ，入出力，論理，設定，テキスト) と，5 つの規模 (非常に大きい，大きい，中くらい，小さい，非常に小さい) に開発者自身が分類し，見積もりのためのデータとして蓄積する．図 2 に蓄積データの例を示す．そして，新たな開発の計画立案工程において，その開発では，どのようなカテゴリ・規模のクラス (関数) がいくつ必要なのかを，概要設計をもとに予測し，プログラム規模，開発時間を回帰分析法を用いて予測する．

3　本授業の内容とプロセスデータ

本節では，PSP の授業実施状況の説明と実施中間結果およびその考察を示す．本授業は 1999 年度に第 1 回を行い，2000 年度は第 2 回である．受講者は，本学の大学院修士課程の学生 5 名である．ちなみに 1 学年の人数は 59 名である．授業開始当初は 8 名 (含む聴講の学部 4 年生) の受講生がいたが 3 名脱落した．

授業の遂行は，概ね教科書 [6] に従い行ったが，以下の点が異なる．

プロジェクト計画立案(PSP0.1)

受講者 海谷 治彦　　日付 2000/10/25
プログラム名 LOCカウンタ　　プログラム番号 3A
講師 海谷 治彦　　言語 C

プログラム規模	計画値	実績値	累積値
ベース(B)	*	0	*
削除(D)	*	0	*
修正(M)	*	0	*
追加(A)	*	86	*
再利用(R)	*	0	0
新規・変更LOC(N)	100	86	185
総LOC(T)	*	86	185
将来の再利用総LOC	*	0	0

工程別時間	計画値	実績値	累積値	累積値(%)
計画立案	11	6	13	5.31
設計	3	14	16	6.53
学習項目決定	0	0	0	0.00
学習	0	0	0	0.00
コーディング	49	81	112	45.71
コンパイル	30	12	31	12.65
テスト	16	24	34	13.88
事後分析	11	32	39	15.92
合計	120	169	245	100

作り込み欠陥	計画値	実績値	累積値	累積値(%)
計画立案	*	0	0	0.0
設計	*	1	1	7.1
学習項目決定	*	0	0	0.0
学習	*	0	0	0.0
コーディング	*	7	13	92.9
コンパイル	*	0	0	0.0
テスト	*	0	0	0.0
開発合計	*	8	14	100

除去欠陥	計画値	実績値	累積値	累積値(%)
計画立案	*	0	0	0.0
設計	*	0	0	0.0
学習項目決定	*	0	0	0.0
学習	*	0	0	0.0
コーディング	*	0	0	0.0
コンパイル	*	3	9	64.3
テスト	*	5	5	35.7
開発合計	*	8	14	100

開発後	*	0	0	*

PSP0.1計画立案スクリプト

工程番号	目的	PSP計画立案プロセスをガイドする
	開始条件	問題記述
		PSP0.1プロジェクト計画概要フォーム
		時間記録ログ
1	プログラム要件	プログラムの要求記述を作るか手に入れる
		要求記述は明確で，曖昧でないことを確実にする
		疑問はすべて解決されてい
2	規模見積もり	プログラム開発で必要とする新規および変更のLOCをできれうだけ上手く見積もる
3	資源見積もり	プログラム開発で必要とする時間を出きるだけ上手く見積もる．
		最も最近に開発したプログラムの累積値(%)をガイドに用いて，開発時間を計画しているプロジェクトの全工程に分配する．
	終了条件	文書化された要求記述
		プログラム規模と開発時間の見積もりデータが記入されたプロジェクト計画概要
		完成した時間記録ログ

図 1: フォーム (上) とスクリプト (下) の例

オブジェクトの規模

オブジェクト数	8					
合計		23.070	0.287		172.000	1702.000
平均		2.884	*		21.500	*
分散(合計/n)			0.036			212.750
標準偏差 = √分散 = σ			0.189			14.586

範囲の中間点の規模 log	範囲		規模	範囲		範囲の中間点の規模
12.24	11.13	13.46	非常に小さい	-14.96	-0.38	-7.67
14.79	13.46	16.26	小さい	-0.38	14.21	6.91
17.88	16.26	19.66	中くらい	14.21	28.79	21.50
21.61	19.66	23.76	大きい	28.79	43.38	36.09
26.12	23.76	28.72	非常に大きい	43.38	36.46	50.67

オブジェクト名	オブジェクトLOC	メソッド数	ln(LOC/メソッド)	(ln LOC-ln LOCavg)^2	LOC/メソッド	(LOC-LOCavg)^2
remove_lastnewline	9	1	2.197	0.074	9.000	156.250
remove_backslashed	14	1	2.639	0.008	14.000	56.250
remove_string	38	1	3.638	0.054	38.000	272.250
remove_comment	53	1	3.970	0.102	53.000	992.250
remove_constchar	13	1	2.565	0.014	13.000	72.250
get_function_name	18	1	2.890	0.000	18.000	12.250
path_name2program_name	16	1	2.773	0.002	16.000	30.250
second_data 4A	11	1	2.398	0.034	11.000	110.250
			0.000	0.000	0.000	0.000
			0.000	0.000	0.000	0.000

図 2: クラス (関数) の規模予測データの例 (テキスト型)

- 表計算ソフトウェアによる簡単なフォーム記入ツールを講師側から提供．ツールの機能は以下の通りである．
  - 開発時間を計るための簡易タイマー
  - フォーム内で二次的に得られるデータの自動計算

  ツールは一開発内で基本的に閉じており，先行開発のデータは自動参照できない．
- 演習は 10A までででなく，9A までとした．演習の遂行ペースは 1 週間に一つ程度である．
- 演習 8A, 9A は PSP2.1 でなく PSP2 で行った．即ち設計図 (OMT や UML 風の設計仕様書) の正式な導入は行わなかった．
- 開発言語は一律，C 言語に統一した．
- コーディング標準は講師側が一律指定した．この標準は，教科書の付録 C に例として提示されていた C++用のものをもとにしている．

まず，受講者が PSP に費やした時間を図 3 に示す．図に示すように，平均して 3 ~ 6 時間程度の時間を PSP に費やしているが，実際には PSP 記録外で行う作業，例えば，問題自体のドメン知識 (主に統計知識) の補完や，PSP 手法自体の学習が必要なため，さらなる時間が必要であった．

次にプロセス改善という言葉から最も直感的に得られるデータとして，時間当りのプログラム作成行数 (生産性) を図 4 示す．図に示す通り特別に生産性が上がっているわけではない．

プロセス改善に関する第 2 のポイントは欠陥除去能力に関してである．図 5 に，学生全員・全課題 (1A-8A) に関する 1 欠陥当りの除去時間を，欠陥作り込み工程と除去工程別に示す．図中の縦軸は作り込み工程を示してお

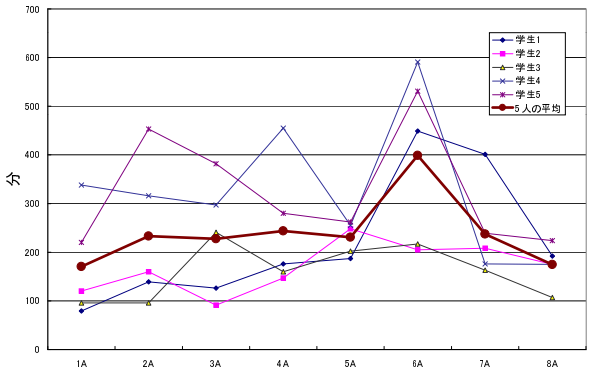

図 3: PSP に費やした時間

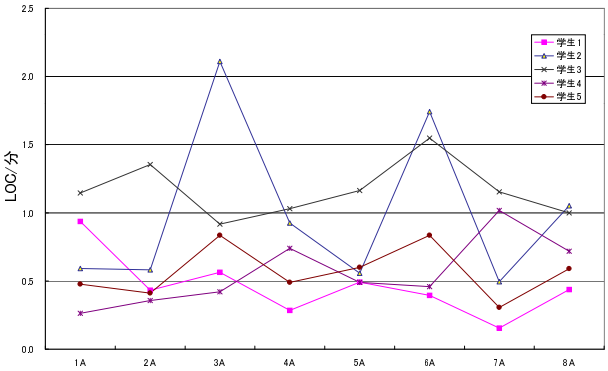

図 4: 単位時間当りのプログラム作成行数 (生産性)

り，横軸は除去工程を示している．例えば，設計で作りこんだ欠陥をコーディングで除去する場合，平均して 4 分かかることになる．コーディングで作り込んだ欠陥に関しては，作り込み・除去工程の距離が離れているほど除去時間がかかるという傾向がある．しかし，設計に関してはその傾向はない．

表 1 に示したように，PSP1 からは Probe 見積もりを行うために，計画工程での比較的綿密なクラス (関数) 設計が必要となり，PSP2 からはレビューを行うため，コンパイラに頼る欠陥除去は減ると思われる．そこで，PSP0(1A-3A)，PSP1(4A-6A)，PSP2(7A-8A) 毎の欠陥作り込み・除去の分布データを図 6 に示す．図中の縦横軸は図 5 と同様だが，それぞれの要素内の数値は，上段が欠陥除去時間の総計 (分) であり，下段が欠陥数 (個) である．例えば，「(a) 1A から 3A の累計」では，コーディングで作り込み，コンパイルで除去する欠陥の総除去時間は 129 分であり，その欠陥数は 64 個である．残念ながら，図 6 の (a), (b), (c) の間には，それほど大きな差異はない．

そこで，欠陥の作り込み工程と除去工程がどの程度離れているかを，図 7 に示すように数値化する．作り込み工程と除去工程が一致する場合は 1 欠陥を 1 点とし，除去工程が 1 つ後ろに下がるにつれ 1 点ずつ加算する．その合計を全欠陥数で正規化したものである．これによって，作り込み・除去工程が離れている欠陥が多いほど，この数値は大きくなる．図 7 の例では，

```
(1*2+2*3+3*16+4*8+4*1+6*2+6*1)/33 =
3.33
```

である．この結果，

PSP レベル	図 7 の数値
PSP0 ((a)1A から 3A)	3.5
PSP1 ((b)4A から 6A)	3.4
PSP2 ((c)7A から 8A)	3.3

となり，若干ながら PSP のレベルアップにつれて距離は縮まっている．

PSP におけるプロセス改善のキーポイントとして，見積もり能力の改善がある．図 8 と図 9 にその経緯を示す．新規・変更 LOC および開発時間に関して，実績値を見積もり値で割った値の推移である．よって，100 を上回った場合，過小見積もりを行ったことになる．理想では，例題を経る毎に 100 に近づくことが望ましいが，双方ともその傾向はない．尚，図 8 の 1A がほぼ全員 0 なのは，PSP0(演習 1A) では LOC を計測しないからである．また，PSP0.1(2A, 3A) では行数の見積もりが指示されているが，その指示内容が「最も良いと判断する方法で開発を予定する新規・変更 LOC を見積もる．」とあるだけなため，見積もりを行えなかった学生もいた．

図 6: 欠陥作り込み・除去個数と時間

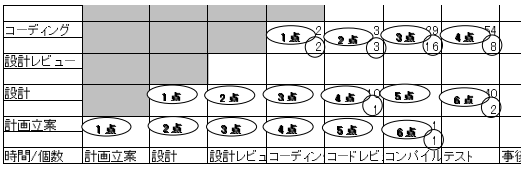

図 7: 作り込みと除去の距離を数値化

作り込み									
開発後									
事後分析									
テスト									
コンパイル									
コードレビュー									
コーディング				1.0	1.0	1.8	8.1		
設計レビュー									
設計		12.5		4.0	10.0	1.0	11.8		60.0
計画立案						1.0			
	計画立案	設計	設計レビュー	コーディング	コードレビュー	コンパイル	テスト	事後分析	開発後

除去

図 5: 1 欠陥当りの除去時間 (分)

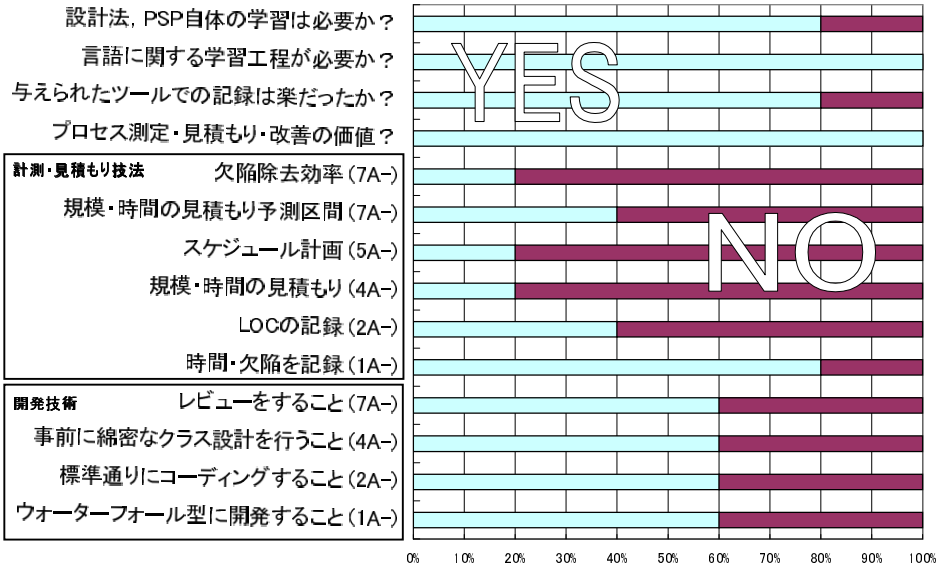

図 10: アンケート

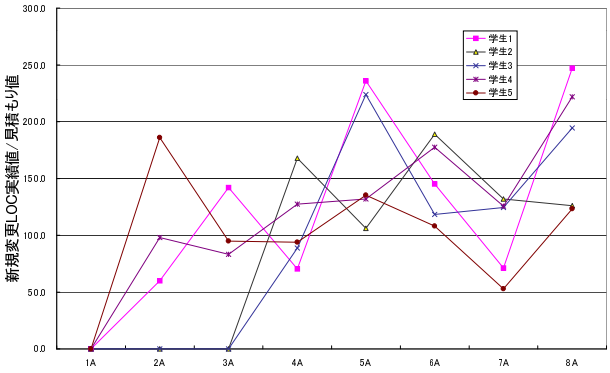

図 8: 見積もりの正確さ (新規変更 LOC 数に関して実績値/見積もり値を計算)

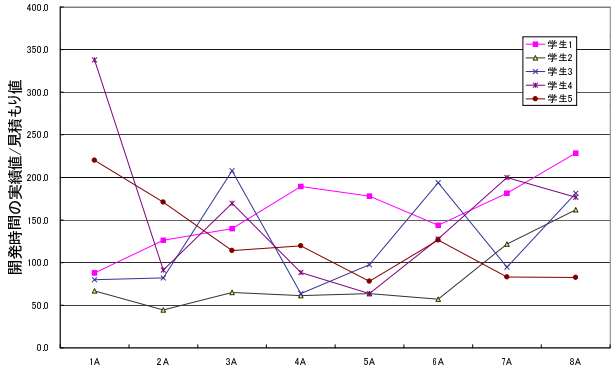

図 9: 見積もりの正確さ (開発時間に関して実績値/見積もり値を計算)

4　大学でのプロセス改善教育のあり方

図 10 にアンケートの結果を示す．プロセス測定・見積もり・改善することに関する価値は，全ての参加者が認識した (4 行目)．しかし，個別の項目に関しては，特に見積もり・計画に関する作業に関する価値に関しては全般に否定的である．これは，見積もり・計画が全く当っていないことが最大の原因だと思われる．しかし，時間・欠陥追跡に関しては，概ね肯定的な意見が得られた．アンケートの最後の 4 項目 (開発技術) の結果から，通常の教科書で説明されている常識的な実践である「レビュー，コーディング標準，事前の設計」などに関する価値を認めた学生の方が若干多かった．残念ながら大学でのソフトウェア開発ではこのような実践を行わない場合が多い．よって，大学などの短い期間の PSP 適用では，開発見積もり・計画には重点を置かず，自分自身のプロセスを客観的に観察することと，教科書的に紹介されている技法の有用性を実感してもらうことに重点を置くべきであると思われる．そして，見積もり・計画法は，その技法の計算練習と割りきって学習してもらうのが良いと思われる．

表計算ソフトウェアによるツールサポートについては概ね好評であった．1999 年の授業では，表計算ソフトウェアによる支援は一切なく，通常のワードプロセッサによって PSP のフォームを作成していたため，受講者の記録等の負担はかなり大きかった．ツール自体のさらなる改善要求として，ソフトウェアのインストール・ウイザードのように，次に入れるべきデータをツール側でできるだけ選別するようにして欲しいという要求もあった．現状のツールは，教科書内に指定してあるフォーム内のデータで，二次的に得られる値を自動計算する程度に過ぎないため，どのセルからデータを埋めるべきか迷う場合があるという意見があった．

1 節で述べたように，学生が PSP を学び始める時点で開発言語・環境の知識を完全に理解していることは少ない．また，実世界の開発においても，言語・環境の変化は年々速くなっているため，言語・環境の学習コストは無視できない．こういった点を鑑みて，PSP の演習を行う場合でも，開発言語・環境に関する習熟度の補正を行うべきであると思われる．この方法の 1 つとして，開発工程の中に明示的に「学習工程」を設け，その作業時間，作業 (学習) 項目の個数などを利用して，プロセス改善と言語・環境理解のデータを分離することが可能かと思われる．

謝辞

本論文をまとめるにあたり，2000 年度情報計測特論第一 [10] 受講者諸君に感謝する．また，PSP に関する多数の知見を与えていただいた久保 宏志 氏，野中 誠 先生をはじめとする PSP ネットワーク [11] および PSP セミナーの皆さんに記して謝辞を示す．尚，本研究の一部は，文部科学省 科学研究費 奨励 (A) 課題番号 12780210 の援助の下に実施された．

参考文献・URL

[1] 大学の理工系学部情報系学科のためのコンピュータサイエンス教育カリキュラム J97. http://www.ipsj.or.jp/katsudou/chosa/J97dist.html , Jan. 2000. 現在.

[2] Timothy C. Lethbridge. What Knowledge Is Important to a Software Professional? *Computer*, Vol. 33, No. 5, pp. 44–50, May 2000.

[3] Computing Curricula 2001. http://www.computer.org/education/cc2001/index.htm , Jan. 2000. 現在.

[4] TSP/PSP ホームページ. http://www.sei.cmu.edu/tsp/, Jan. 2001. 現在.

[5] W. S. Humphrey. *A Discipline for Software Engineering*. Addison-Wesley, 1995. The Complete PSP^{SM} Book.

[6] Watts S.Humphrey 著, ソフトウェア品質経営研究会訳. パーソナルソフトウェアプロセス技法 – 能力向上の決め手. 共立出版, May 1999.

[7] W. S. Humphrey. *Introduction to the Personal Software Process*. Addison-Wesley, 1997.

[8] Ralph F. Grove. Using the personal software process to motivate good programming practices. In *Annual Joint Conference Integrating Technology into Computer Science Education*, pp. 98–101, Aug. 1998.

[9] Susan K. Lisack. The Personal Software Process in the Classroom: Student Reactions (An Experience Report). In *Proceedings of the Thirteenth Conference on Software Engineering Education & Training*, 1998.

[10] 情報計測特論第 1 ホームページ. http://dotcom.cs.shinshu-u.ac.jp/psp/, Jan. 2001. 現在.

[11] PSP ネットワークホームページ. http://www.azuma.mgmt.waseda.ac.jp/psp/, Jan. 2001. 現在.