UNIXの使用経験

株式会社　ソフトウェア・リサーチ・アソシエイツ

栗原正利

１．はじめに

ＳＲＡにUNIXシステムが導入されてから，ほぼ３年の時間が経過した。現在，日本国内におけるUNIXユーザの数は500程度だが，そのほとんどは大学・研究所である。

２．導入時の問題点

図１は，ＳＲＡにおけるUNIXの歴史を簡単に要約したものである。

使用しているCPUは，VAX-11／780であり，これはBell Labs版でなく，UC Berkeley版のUNIXを導入するために選択された。ハードウェアは80年６月に設置し，UNIXは，最初３BSD（Berkeley版のVer.３）を80年８月にインストールした。その後，81年３月に4BSDへと変更した。

当初の計画では，最初の１年間は実験的にのみ利用することを考えていたので，主メモリおよびディスクの容量はかなり小さく，端末の数も８台と，小規模な構成で

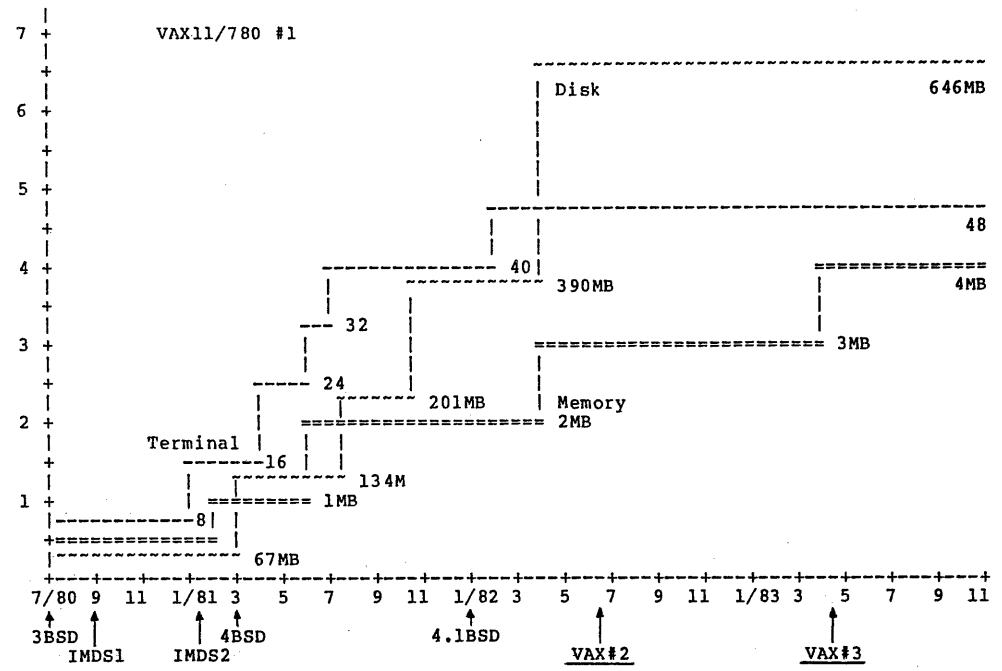


図１　ＳＲＡにおけるUNIXの歴史

あった。導入後半年を経過した時点で，かなり実用化の自信が得られ，on UNIXプロジェクトの数も当初の予想をはるかに上まわり，それにあわせ，予定より少し早いペースでハードウェアの増強が行なわれたが，需要に追いつけない状態であった。

8月のシステム・インストール時点には，社内スタッフがまったく未経験であったことも考慮して，UC Berkeleyの大学院生を1人，パート・タイム・サポーターとして採用し，夏休み期間を利用して，約1ヶ月間の技術指導を依頼した。このことは，「立ち上がり」の効率を上げるうえでも，また，社内のシステム・プログラミング・スタッフに対する知的刺激という意味でも，大成功であったと評価している。

一応のインストールが完了し，最初の実験プロジェクトであるIMDS 1 （対話型モジュール開発システム）が軌道に乗った80年秋から，全社員を対象に，UNIX入門のための講習会を開催した。そのカリキュラムは，表1のとおりである。

表1　UNIX トレーニングのカリキュラム

No.	内　　容	時　間
1	UNIX概説	2
2	Editor　(A)	2
3	Editor　(B)	2
4	Shell	2
5	Tools　(A)	2
6	Tools　(B)	2
7	Testbed	2

この講習会は，6ヶ月間にわたって計8回行なわれ，総受講者数は200人にのぼったが，教育効果としては，単なる社内技術PRの域を出ず，失敗であったといえよう。今後の教育は，この失敗を生かして，より実務的な訓練を効果的に行なうべく，特定の技術的課題をもった少数の人間を対象としたOJT方式に切りかえることとした。UNIXを有効に活用するためには，それぞれのプロジェクトの内容や形態に合ったソフトウェア開発環境を整備すべきであり，そのための何らかの課題をもってトレーニングに望むことは，教育効果および現場に適した環境整備に，大いに有効であると考えたからである。

P.J.デニングが指摘しているように（参考文献1），UNIXにおけるプログラミングのスタイルは，旧来のそれとは根本的に異なるので，ソフトウェア開発の現場にUNIXを導入するさいには，プログラマやプロジェクト・マネージャとのあいだで，一種の「文化的」または「精神的」な摩擦が生ずる。

すでに何年かの実務経験を経て，一定の思考習慣や仕事のスタイルを身につけた人々の中にUNIXスタイルの新しいプログラミングのやり方を浸透させるには，少し時間がかかりそうである。あせらずに，気長に，ヴァラエティに富んだ教育訓練をくりかえして行くほかはないようだ。

実際のプロジェクトをUNIXにのせるさいの一番の大きな問題点は，個々のプロジェクトに特有の作業条件(作業内容，プロジェクト構成，期間，プロジェクトの大きさ，等々）に合わせて，それにもっとも適した環境を用

		On-line	Batch	Total
Number of Programs	(FORTRAN)	173	158	331
Total Size	(NCSS)	15,670	17,500	33,170
Average Size of Programs	(NCSS)	90	110	100
Period of Development	(MONTHS)	3	3	-
Number of Programmers		7	9	16
Number of Managers		1	1	2
Number of Terminals		5	-	-

Programming Environment

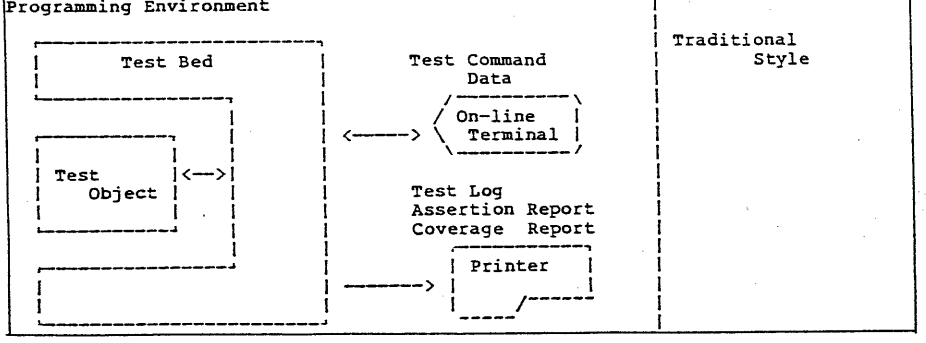


図2　プロジェクトの概要

意してやる必要があるということである。

もともとのUNIX自体が提供する環境は，かなりスマートなシステム・プログラマ向きのそれであって，「平均的な」アプリケーション・プログラマには使いこなしにくく，また自分でその環境を整備することもむずかしい。

そこで，われわれの場合には，専任のUNIXシステム・プログラミング・スタッフが，各プロジェクトのリーダと相談しながら，プロジェクトごとの環境を，そのスタート時に整備するという方式をとった。これもまた，一種のOJTであり，その過程を通じて，現場サイドに少しずつ「わかる」人間をふやして行くことが重要だと考えている。

3．モジュール・テスト・ベッドの経験

最初にUNIXにのせた実務プロジェクトは，プロセス制御アプリケーション・プログラムの開発であった。モジュール・テスト・ベッドまたは対話型モジュール開発システム（IMDS）と呼ばれる環境が，そのために整備された（参考文献２，３）。

図２は，そのプロジェクトの概要を示す。全体で 331本のモジュールのうち，約半数がUNIX上で，残りの半数は旧来のプログラミング環境で開発された。

このプロジェクトの発足時には，まだUNIXの端末数も少なく，また，プロジェクト・メンバーも対話型環境に慣れていなかったので，オリジナル・コーディングまでは，これまでと同じ紙とエンピツで作業し，ソース・プログラムをいったんカードにパンチしたあと，MTでUNIX環境に持ってきて，デバッグおよびテストだけを対話型で行なうという方式をとった。

結果は図３に示すとおりであるが，UNIXを利用したオンライン対話型モジュール開発による「生産性」の向上がはっきりと示されている。

まず，モジュールの開発段階であるが，UNIX上ではほぼ予定通りの開発工数で作業が終了しているのに対し，旧来の環境では，例によって２倍近くの遅れが発生し，人月あたりの生産性はUNIXの場合とくらべると半分以下に落ちている。

またUNIX上では，テスト・カバレージ計測ツールを用いて，Ｃ１カバレージ90%のラインでモジュール・テストの完全性が保証されているが，旧来の環境下ではそうした管理はなされていない。おそらく，計測すればＣ１≦50%程度であろう。

次に統合テスト段階をみると，この段階で発見されたプログラミング・エラーの個数が，それぞれの環境におけるモジュール開発の信頼性を示すものとすれば，やはりUNIX環境のほうが，２倍以上すぐれている。当然のことながら，旧来の環境では，より多くのエラーを修正するために，余分の労力をこの段階でも必要としている。

	On-line	Batch
Programming Period	3	5
Working Time (MAN-MONTH)	22.3	56.6 (50.2+6.4)
Productivity (NCSS/W-TIME)	706	309
Coverage (%)	90	-
Integration Test Period	2	4
Number of Programming Errors	49	122
Working Time (MAN-MONTH)	13.7	32.9
Total Size/Working Time	1150	532
Programming + Integration Test Working Time (MAN-MONTH)	36	89.5
Productivity (NCSS/W-TIME)	438	196

図３　オンライン開発とバッチ開発の比較

以下を総合して，人月あたりの開発ステップ数という単純な指標でみたUNIX環境のほうが、約 2.5倍も効率的であることがわかる。

ところで，プロジェクト・メンバーの作業ぶりを個人別に眺めてみると，そこには，かなり大きな個人差がある。このプロジェクトの場合では，UNIX環境の上でのベスト・プログラマとワースト・プログラマとの生産性の比較は約３倍であった（しかし，ワーストでも旧来のバッチ環境と比べれば，２倍近くの生産性をあげている）。その差が出てくる原因は，できるプログラマの場合には，UNIXのもつさまざまな機能をフルに活用して仕事を効率化しているのに対し，平均的（あるいは平均以下の）プログラマは，与えられたプロジェクトの環境（すなわち，システム・プログラマの手で用意されたプロジェク

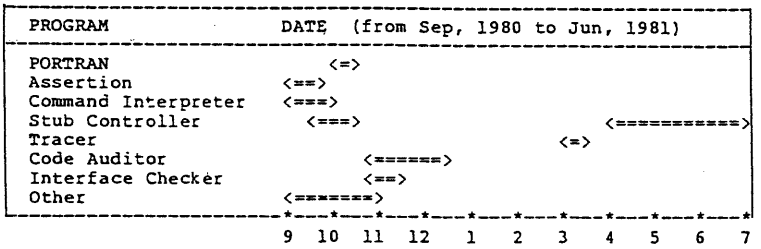
ト用の特殊コマンド）だけを慢然と利用しているだけでしかないというあたりにある。

その意味からすれば，UNIXもまた，他のあらゆるソフトウェア・ツールや開発技法と同じく，それをうまく使いこなせるか否かによって，プログラマのランク付を可能にするメディアである。

ラムすれば数十ないし数百ステップに相当するのだから，その比率で換算すると，さらに驚異的な数字になるだろう。

こうした高生産性は，UNIXに固有の対話型プログラミング・スタイルに起因している。

UNIXでは，すべてのユーティリティ・ツールのソー



PROGRAM	STEP	#FUNC	MAN-MONTH	PRODUCTIVITY
PORTRAN	845		.5	1690
Assertion	585	7	.5	1170
Command Interpreter	1782	11	1.0	1782
Stub Controller	2803	50	4.0	700
Tracer	364	5	.2	1820
Code Auditor	2109	23	2.0	1054
Interface Checker	1175	21	1.0	1175
Other	1129		1.0	1129
TOTAL	10792		10.2	975

図4　モジュール・テスト・ベット開発の概要

4．システム・プログラミング

図4は，前述のモジュール・テスト・ベッドの環境開発におけるシステムの各要素別の詳細を示したものである。

モジュール・テスト・ベッドの第1版は，80年9月～10月に開発され，すぐ実用に供されたが，その後，段階的に改良が加えられ，現在もまだ継続中である。この図は，81年6月末時点までの総括を示している。

図3のアプリケーション・モジュール開発の数字と比較して，システム・プログラミング作業の効率が，きわめて高いことに気付かれるだろう。しかも，前者が，プログラミング以降テスト完了までの部分的な工程でしかないのに対し，後者は，テスト・ベッドの要求定義からテストまでの開発フェイズ全体をカバーしているのである。

このことは，プログラミング・ツールとしてのUNIXの持つ強力なパワーをはっきりと示している。

図4のステップ数は，Cプログラム，Shellコマンド，Yacc／Lexへの入力等，システム・プログラマの書いた（キーインした）ソース・ステップ数をあらわしているが，1行のShellコマンドは，ふつうにその機能をプログラム・プログラムがディスクに格納されており，オンラインでいつでも取り出せるようになっていて，システム・プログラマは，自分が何か新しいツールを開発するさいに，それらをフルに利用することができる。

すなわち，いま，ある機能Aをもつツールを作ろうとしたとき，機能Aが副次的な機能BおよびCから構成されているものとしよう。機能BはShellコマンドXに含まれており，一方機能Cはかれが過去に作ったツールYに含まれている。その場合，システム・プログラマは，XおよびYのソース・プログラムをエディタによって適当に切り貼りすることで，自分が望む機能Aをもつツールのプログラミングに代用しうるのである。

すべてのプログラムが「使い捨て」でなく，このように過去のプログラム遺産をうまく再利用できることが，UNIXのもつパワーの秘密である。新たにプログラムを書かずに，プログラムを書いたと同じ効果を達成することこそが，ソフトウェア開発における生産向上の唯一の決め手なのである。

もちろん，そうしたことが容易に行なえるかどうかは，ユーザ・インタフェイスとしてのコマンド・インタプリタおよびエディタの機能が充実しているか否かにかかっている。

Berkeley版のUNIXは，その意味で，もともとのBell

Labs版よりも，数倍高い性能をもっているといっていいだろう。

Bell版のShell コマンド・インタープリタのサイズは，C言語のソースで4 Kステップであるが，Berkeley版のC-Shellは11.5Kステップと約3倍の大きさであり，対話型ジョブ・コントロール等のいくつかのきわめて便利な機能を提供してくれる。

一方，エディタについていえば，Bell版のテキスト・エディタedのコマンドは30種，ソース・ステップ数1.8Kであるのに対し，Berkeley版のスクリーン・エディタviのコマンドは86種×41オプション，ソース・ステップ数は20Kをこえ，実にOSのKernel部よりも大きい。さらにいえば，このviの中では，すべてのShellコマンドがエディタの「サブコマンド」として利用できるようになっている。すなわち，エディタがOSの管理下にあるのではなく，OSがエディタの一部になっているのである。

こうした革命的な発展に直接触れること，しかもそれが自分とほぼ同世代のプログラマの仕事だと知ることは，システム・プログラマにとって，この上ない知的刺激になるであろう。

5．なぜUNIXか

われわれがUNIXシステムを導入した理由は，要約すれば次の3つであった。

第1に，国内外における先進的ソフトウェア技術をトランスファするためのメディアとしてUNIXがきわめて適切であると考えられたこと。この点はすでにBerkeleyの「気狂いプログラマ」たちの作風に接したことの衝撃だけでも，十分に効果があったと考えられる。

第2は，社内における技術の蓄積や交流を促進すること。この点は今後の課題であるが，最初のプロジェクトの成功に触発された形で，UNIXの利用が予定以上のハイ・ペースで拡大していることから，ほぼ問題なく進展するだろうと見ている。

第3は，ややマンネリ化したソフトウェア開発組織を新しい形に改造すること。これはそう急にはむずかしいだろうが，5ヶ年計画で全社の作業をオンライン化し，在宅勤務の可能性までを含めて新しい仕事のスタイルとそれに合った組織作りを計画中である。