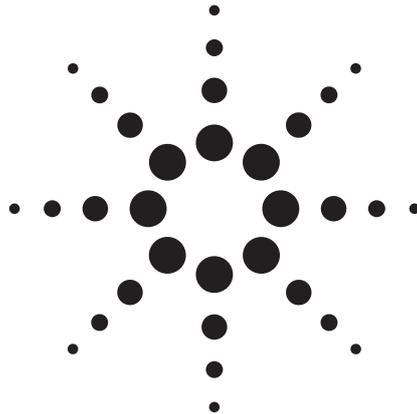


テスト・システム 開発ガイド

テスト・システムの
ソフトウェア・アーキテクチャ

Application Note 1465-4



このアプリケーション・ノートは、「テスト・システム開発ガイド」シリーズのアプリケーション・ノートです。信頼性の高いテスト・システムのデザイン、スループットの向上に役立ちます。

ここにはソフトウェアの方向性を決定する際に役立つ情報が数多く掲載されています。ソフトウェア要件の収集／文書化からデザインを再利用する上での注意事項まで、ソフトウェア開発のプロセス全体について説明しています。

本シリーズのアプリケーション・ノートのリストは、19ページにあります。

目次

はじめに	2
ソフトウェア要件の収集／文書化	3
測定器のプログラミング／制御	5
テスト・データの収集／保存	6
ユーザ・インタフェースの設計	9
開発環境の選択	10
オープン・スタンダードへの取組み	12
テスト・シーケンスの開発	14
ソフトウェアの再利用	15
まとめ	17
用語解説	18
関連カタログ	19



Agilent Technologies

はじめに

このアプリケーション・ノートでは、ソフトウェア・コンポーネントの設計、開発に必要な情報を提供しています(図1を参照)。

ソフトウェア要件の収集/文書化:ソフトウェア要件を収集/文書化する前に、ハードウェアの仕様を決定します。その仕様に基づき、研究開発や製造のチームと協力して、ソフトウェア要件仕様書(SRS)を作成するのに必要な情報の収集に取り掛かります。

測定器のプログラミング/制御:測定器の制御は、独自のプログラミング環境からオープンな環境へと急速に進化しつつあります。こうした傾向は、測定器を制御するソフトウェアやドライバはもちろん、PCと測定器を接続するハードウェアにも影響を及ぼしています。

テスト・データの収集/保存:テスト・システムの品質は、適切な時に適切なデータを入手できるかどうかにかかっています。

ユーザ・インタフェースの設計:テスト・システムの最も重要で見逃しやすい1つが、グラフィカル・ユーザ・インタフェース(GUI)です。GUIは、テスト・エンジニア、オペレータがソフトウェアを使用する際の作業効率を決定します。

開発環境の選択:選択するソフトウェア環境やツールは、テスト・システムのコスト全体に大きな影響を与えます。ソフトウェア環境を選択する際には、ソフトウェアの購入価格だけでなく、それ以外のことも考慮する必要があります。さらに、サポート/保守コストはもちろん、ソフトウェアの習得/使用の容易さ、他の言語、デバイス、エンタープライズ・アプリケーションへの接続の容易さも考慮する必要があります。テスト・システムの耐用期間にわたるソフトウェアのサポート/保守コストだけでも、ハードウェア・コストを上回ってしまう可能性があります。

オープン・スタンダードへの取組み:今日では、業界は閉鎖的な独自の環境から脱皮する傾向にあります。ますます多くの人が、テスト・システム開発プロジェクトのプラットフォームとして、オープンな業界標準の開発環境を取り入れるようになって来ています。開発環境の選択により、将来必要となる柔軟性と機能が決定します。

テスト・シーケンスの開発:テスト・エグゼクティブは、テストを既定の順番で実行するように設計されたアプリケーションです。テスト機器を使用している93%のテスト・システム開発者のうちの約37%が、市販のテスト・エグゼクティブを用いているのに対して、残りの56%は「自社開発」のテスト・エグゼクティブを使用しています。

ソフトウェアの再利用:コードの再利用できれば、新しいプロジェクトを開始する度に、ソフトウェアを作り直す必要がなくなります。さらに、最良のアイデア、最良の手順、ソフトウェアに関する会社のナレッジ・ベースを確立することができます。こうしたナレッジ・ベースは、会社の製品テスト部門に画一性と一貫性をもたらします。

このアプリケーション・ノートでは、上述のようなテスト・システムのソフトウェア・アーキテクチャについて詳しく説明しています。詳細については、本書に掲載されている文献を参照してください。

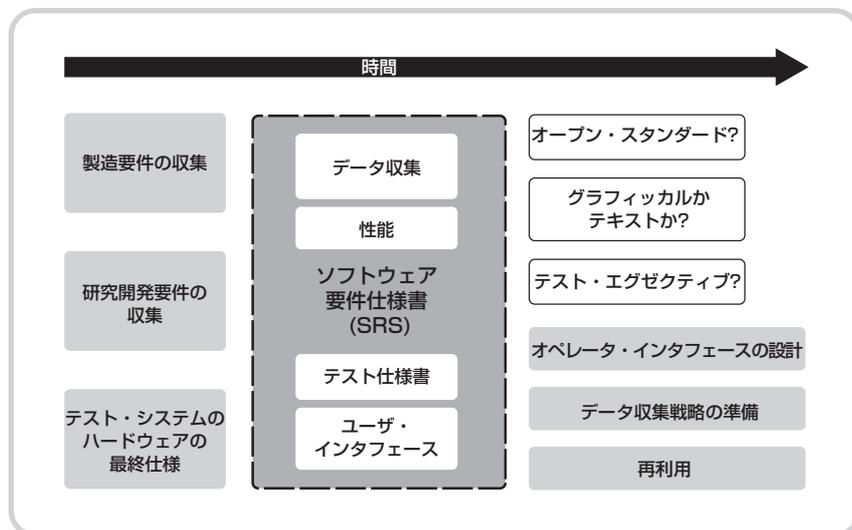


図1. テスト・システムのソフトウェア開発プロセスの概要

ソフトウェア要件の収集／文書化

ソフトウェア要件仕様書(SRS)¹とは、テスト・システムのソフトウェア機能とソフトウェアの外部インタフェース、性能要件、システム属性、設計上の制約に関する情報を優先順位を付けて記載した文書です。一般に、一部の要件「必須」は絶対不可欠ですが、その他の「要求」は時間には代えがたい場合があります(例えば、プロジェクトの期日に間に合わせるため)。

IEEE協会は、SRSの中で扱うべき情報として以下を規定しています。²

- 機能：ソフトウェアは何をするのか?
- 外部インタフェース：ソフトウェアは人、システムのハードウェア、その他のハードウェアやソフトウェアとどのように対話するのか?
- 性能：様々なソフトウェア機能の速度、可用性、応答時間、回復時間は?
- 属性：ポータビリティ、正確さ、保守性、セキュリティに関する考慮は?
- 設計上の制約：準拠する必要がある業界標準は? 特定の言語を使用する必要があるのか? データベースの品質、リソースの制限、動作環境に関する社内の方針は?

SRSには、処理方法ではなく、ソフトウェアが実行すべき処理内容を記載するのが理想です。つまり、ソフトウェアは、測定器、コンピュータ・モニタ、その他のコンポーネントなどの外部リソースを制御する「ブラック・ボックス」と見なすことができます(図2を参照)。

SRSには、要件が外部に公表されている場合にのみ実装に関する詳細を含めます。例えば、システムの一部を特定のプログラミング言語で実装することを求めることもできます。

優れたSRSには、次の質問に対する答えが示されていないければなりません。

1. 被試験デバイス(DUT)の試験にどのような測定／テストが必要か?
2. 使用可能な測定器／デバイスがあると仮定した場合、測定／テストはどのように実行されるのか?
3. どのような種類のデータを収集する必要があるのか?
4. データの保存場所は?
5. どのような外部的な制約(すなわち、性能および時間の仕様)があるのか?

6. オペレータ、テスト・エンジニアはどのようにしてソフトウェアと対話するのか?

製品開発のライフサイクルで、研究開発部門は正式のテスト要件リストをテスト開発部門に提供する必要があります。システム要件仕様書は、プロジェクト要件仕様書とも呼ばれ、システム全体について言及しているため、ソフトウェア要件仕様書とは異なります。さらに、製造部門には、安全規格などの独自の要件があります。テスト・システムのハードウェア要件を決定し、ソフトウェア要件仕様書の基礎ともなる、研究開発仕様書と製造仕様書を組み合わせたものです。

テスト・システムのハードウェア仕様が流動的な状態でソフトウェアを設計すると、ソフトウェアのリワーク／再設計が必要になる場合があることを考慮しておく必要があります。

1 ERSまたは単に「要件」と呼ばれることもあります。

2 詳細については、IEEE Webサイト(<http://standards.ieee.org>)に掲載されているIEEE規格830-1998 “Recommended Practice for Software Requirements Specifications”およびIEEE規格1233-1998 “Guide for Developing of System Requirements Specifications”を参照してください。

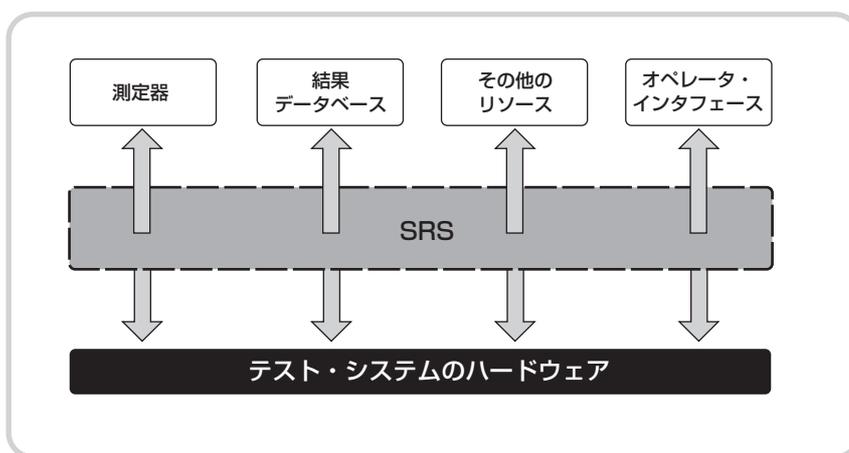


図2. SRSの範囲

図3に、SRSのテンプレートと要件のサンプルを示します。テンプレートに示されているように、SRSは単なる要件ではありません。SRSには、ソフトウェアがどのような処理を実行するように作られているかを定義すると共に、使用する用語の定義も行います。外的制約条件と、使用可能な外部リソースを文書化します。ユーザの詳細と各ユーザ・クラスの動作モードを記述します。最後に、付録と索引を付けます。これらの作業が終わると、特定の要件を記述する準備が整います。

要件のサンプル(テスト・シーケンスのユーザ・インタフェース)は、機能別に分けられた大量の要件の中から抜粋したものです。「必須」および「高要求」という用語は、要件の相対的な重要度をランク付けするための方法の1つです。機能、プログラム・モード、または要件部分の検索を容易にするその他の分類体系に基づいて、要件を管理しやすい階層に分けることができます。

IEEEでは、要件は正確で、一義的で、完全で、一貫性があり、重要度がランク付けされ、検証/変更/トレース可能でなければならないとしています。

テンプレートはこうした目標の多くを満たしていますが、要求をすべてを満たすためにはさらに若干の手続きが必要です。2箇所以上の要件を参照している場合は、要件が変更されたときにどこか他に文書内で修正すべきところがあるか分かるように、固有の番号(例えば、3.4.3)を使って相互参照する必要があります。

テスト・プログラムを期待通りに動作させるためには、記述されている各要件が検証可能で、しかも一義的でなければなりません。SRSを書く際には、可能な限りシステム要件仕様書を参照してください。これは、バックワード・トレーサビリティと呼ばれ、特定の要件が盛り込まれていて、単なる制限事項ではないことを説明するのに有効です。

SRSには、どのようなテスト・リソース(測定器)が必要か(例えば、電圧計、スイッチ、コンピュータ・モニタなどのタイプ)、工場リソースが必要か(例えば、測定データベース)を記述する必要があります。さらに、SRSには、データの収集方法、ユーザ・インタフェースの要件、性能上の制約、さらに最も重要なことですが、特定のDUTテスト条件を定義する必要があります。例えば、特定の抵抗測定を実行しな

ければならない場合、Agilent 34401Aマルチメータを持っていることが分かっている場合は、SRSで4端子測定を指定することによって(適切なスイッチング経路の記述を含む)、DUT上のピンへのアクセスを保証します。

テスト・システムのソフトウェア・ユーザ・インタフェース要件を正確に記述するためには、テスト・システムのユーザごと(例えば、オペレータ、テスト・エンジニア、マネージャなど)に固有の使用例を作成する必要があります。使用例とは、ユーザのソフトウェアとの対話を記述するシナリオです。

正確に記述されている要件仕様書を前もって作成することに時間をかければ、開発プロセスの後半で時間を節約することができます。SRSプロセスは、プロジェクトの範囲について考えることを強いるだけでなく、ソフトウェアの不明な領域を確認するのにも有効です。このため、何が本当に必要だったのか分からなくなってしまったためにソフトウェアを書き直して再テストするというときに時間を費やすことが少なくなります。SRSが正確に記述されていれば、外注化も容易になり、リワークも必要がなくなります。

SRSテンプレートのサンプル

目次

- 1 はじめに
 - 1.1 目的
 - 1.2 範囲
 - 1.3 定義、頭文字、省略名
 - 1.4 参照
 - 1.5 概要
- 2 総論
 - 2.1 製品の展望
 - 2.2 製品の機能
 - 2.3 ユーザの特性
 - 2.4 制約
 - 2.5 前提条件と依存関係
- 3 特別な要件
- 付録
- 索引

要件のサンプル

- 3.4 ユーザ・インタフェースの機能：
- 3.4.1 (必須)UIはユーザによるシーケンスの作成、変更、実行、デバッグを可能にする。
 - 3.4.2 (必須)UIはユーザによるシーケンス実行結果のデータの表示/エクスポート、ロード/ストアを可能にする。
 - 3.4.3 (必須)UIはシーケンスを階層形式で表示し、階層を表示/非表示にして、シーケンス内部の詳細を表示/非表示することもできる。
 - 3.4.4 (高要求)UIはメイン・シーケンス階層とは別に、共有(数箇所で使用される)シーケンスを表示することができる。
 - 3.4.5 (高要求)UIは、グラフィカル・アイコンを使って、シーケンス・アイテムの状態の変化を表示する。

図3. SRSのテンプレートと要件

測定器のプログラミング/制御

テスト・システムのアーキテクチャを設計する場合は、PCと測定器との通信方法を考える必要があります。2つの最も重要な要素は、1) PCを他の測定器に物理的に接続する方法と、2) 他の測定器との通信や制御に使用するソフトウェアです。

コンピュータと測定器の物理的な接続

数十年の間、 GPIBとしてよく知られているIEEE-488バスが、テスト機器、コンピュータ、測定器の接続に使用されてきました。今でも GPIBが一般的な機器インタフェースであることには変わりはありませんが、USBやLANの方がコスト・パフォーマンスの高いソリューションとなりました(表1を参照)。

USBは、システム内の測定器の数が少なく、迅速かつ簡単なインタフェースのセットアップが求められる場合に最適です。

USB 2.0およびイーサネット・ベースのLANは、データ・スループット性能、コスト、リモート・アクセス、システムの組立の容易さが最優先事項となる使用に適しています。USB 2.0とイーサネット・ベースのLANからいずれかを選択するのであれば、ほとんどの人は、柔軟性の高さおよびリモート・システムへのアクセス/制御機能から、LANを選択します。さらに、LAN

の性能はUSBと同等ですが、(USBにはない)脱落しにくいコネクタが採用され、無線動作も可能です。

I/Oソフトウェア・レイヤ

コンピュータと測定器を物理的に接続する方法が決定したら、測定器の制御や通信に使用するI/Oソフトウェアを決定する必要があります(図4を参照)。

I/Oソフトウェアは、ソフトウェアと測定器の物理インタフェースの間にあるソフトウェアの層です。2つの選択肢があります。測定器に直接書き込むか(Direct I/O)、機器ドライバを使用することができます。標準の機器ドライバは使いやすいので一般的ですが、測定器の機能の一部しか利用できません。

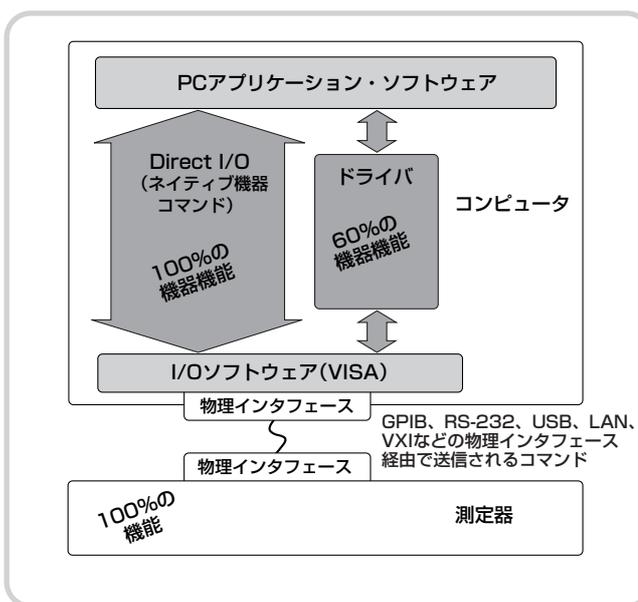


図4. I/Oソフトウェア・レイヤ

表1. GPIB、USB、LANの利点と欠点

インタフェース	理論上のインタフェース速度	利点	欠点
GPIB	<ul style="list-style-type: none"> 8MB/sの伝送速度 	<ul style="list-style-type: none"> テスト機器のユビキタス・インタフェース 全ブロック・サイズのスループットを最大限に高める 低コスト 	<ul style="list-style-type: none"> 拡張スロットが必要 PC筐体を開けてカードをインストールする必要がある 比較的高価 コンピュータと測定器の間に使用できるケーブルの長さ制限がある
USB	<ul style="list-style-type: none"> USB 1.1 12 Mb/s USB 2.0 480 Mb/s 	<ul style="list-style-type: none"> 迅速かつ簡単なセットアップ 低コスト 優れたデータ・スループット性能 	<ul style="list-style-type: none"> Windows® NTで動作しない 配備済みのほとんどの測定器で使用できない
LAN	<ul style="list-style-type: none"> 10/100/1000Mb/sの伝送速度 	<ul style="list-style-type: none"> 優れたデータ・スループット性能 低コスト リモート・アクセスによるリモートからのシステムの制御が簡単 	<ul style="list-style-type: none"> セットアップにはLANに関する知識が必要 配備済みのほとんどの測定器で使用できない

それでは、どのようにして決定するのでしょうか?以下に考慮すべき要素をいくつか示します。

以下のような場合は、Direct I/Oを使用します。

- 使用可能なドライバがサポートしていない測定器機能(測定器の機能の残りの40%~約80%)を使用する必要がある。この場合は、Direct I/Oと機器ドライバの組み合わせを使用することができます。一部のドライバは、このような状況に対応したDirect I/O接続を提供しているため、簡単になります。
- 測定器のプログラミングの経験がある、またはプログラミングの専門家にアクセスできる。
- 最大のシステム処理速度が必要である。
- システム内の測定器の正確な構成を制御する必要がある。
- 既存システムのSCPIベースのコードが大量にある。

以下のような場合は、機器ドライバを使用します。

- 開発環境およびI/Oソフトウェアで動作し、使用したい測定器機能のほとんどをサポートするドライバが使用可能である。
- ドライバにより提供される測定器機能の分かりやすい階層構造によって得られる使い勝手のよさが必要である。

- コードの開発/保守プロセスを簡略化したい。
- 測定器を交換する必要がある場合に、システムの保守を簡略化する必要がある。
- 開発時間が最も重要である。

機器ドライバを選択する場合は、業界標準のIVI-COM(Component Object Model)ドライバ³とVisual Studio®.NET対応の開発環境(Agilent T&MProgrammers Toolkit)を併用することを検討してください。IVI-COMドライバには以下のような利点があります。

1. 一般的なPC言語とほとんどの電子計測言語で動作する。
2. 最も一般的なタイプのI/Oを使用する。
3. 最新の.NETテクノロジーを使用できる。

IVI-COMドライバをテスト・システムのアーキテクチャの開発に使用することにより、時間が節約され、ハードウェアとソフトウェアの互換性も高まります。また、ソフトウェアの保守が容易になり、将来の拡張性も高まります。

テスト・データの収集/保存

データ収集では、テスト・システムの動作とテスト対象のデバイスに関する重要な情報の識別/入手/収集/フォーマット/配布を扱います(図5を参照)。品質データの収集は、製造/テスト・プロセスを制御するための基礎であり、製造テスト・エンジニアの最終目的でもあります。品質データは、組織全体の数多くの部署をサポートすると共に、開発ライフサイクル全体を通して製品をサポートするために用いることもできます。

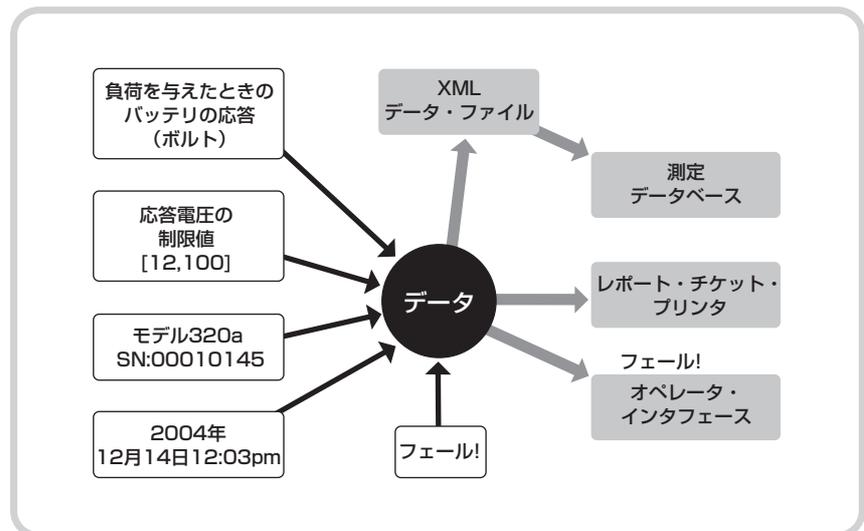


図5. データ収集プロセスの概要

- 3 IVI-Componentドライバは、Microsoft®のコンポーネント・オブジェクト・モデル(COM)をベースとしています。IVI-C(NI)ドライバは、Cのdllをベースとしています。
- 4 Agilent Technologiesが最近実施した調査では、Visual Studio .NETを使用した場合、テスト・プログラマは、テスト・プログラムの開発時間の20%~30%の短縮を実現していることが分かりました。

さらにテスト・データは、テスト・シーケンスの評価、規制への適合、性能の文書化、DUTのトレーサビリティに用いることもできます。こうした用途を考えると、研究開発または製造の人間が求める以上のデータを必要に応じて収集する必要があります。

記録されているデータは、外部からのデータ要求以外にも、テスト・シーケンスをデバッグにも用いることができます。デバッグとは、テスト・シーケンスの動作を遅くしたり、変更して、不具合箇所を発見することです。このようなデバッグの操作では、通常の実行で確認される不具合が明らかにならない場合があります(逆の場合もあります)。診断テスト・ソフトウェアとそれに対応するDUTの負担を軽減する1つの方法は、デバッグに必要なデータを常に収集することです。追加データの収集は、テスト・ソフトウェアの性能と時間に影響を与えるため、これらを比較検討する必要があります。

標準的なタイプのデータ(例えば、テストのリミット値、測定値、合格判定)と同様に重要なのが、コンテキスト・データです。コンテキスト・データは、DUTのテスト環境に関連するすべての情報が含まれます。これには、テスト・システムの構成、ソフトウェアのバージョン、ドライバのバージョン、その他の要素が含まれます。

記録する変数が多いほど、デバッグ中に解析できる相関ポイントの数が多くなります。例えば、ある特定の製造テスト条件では、DUTは午後には不合格になってしまいます。テスト・エンジニアは、日時と不良発生時間を相関させて、その情報を用いてDUTの光コンポーネントをより詳しく調べることができました。

太陽の光がその日の一定の時間にそのコンポーネントを直射したために、コンポーネントによりコンデンサが帯電し、テストに不合格になってしまったことがわかりました。DUTは、温度変化や相対湿度が原因で不合格になる場合があります。コンテキスト情報と測定条件を収集することで、多くの情報が得られます。

データの記述やフォーマットがテスト・システムの動作に影響を与えないようにします。今日のPCは、与えられたファイルまたはネットワークI/Oコマンドに要する時間に大きな影響を及ぼす可能性のある各種キャッシュ・テクノロジーを採用しています。データをキャッシュに入れるのに要する時間がテストを実行する度に異なる場合は、テスト結果は一貫性のないものになってしまいます。そのため、DUTのテストが終わるまでデータをRAMに保存して、その後フォーマットして、データを伝送します。

データは、理解できない限り役に立ちません。有効なデータの特長を以下に示します。

- **識別可能**：データを取り巻く状況およびそれが収集された条件を識別するための情報。

- **検索可能**：一意に識別可能な正規構造またはフィールドにより、スクリプトまたはソフトウェア・ツールが複数のレコードまたはデータセットから簡単に確認/比較できるようにします。

- **変換可能**：生データを解釈し、表示する必要があります(意味のある情報がゴールです)。すなわち、ソフトウェア・アルゴリズムはデータの一部または全フィールドの処理を実行し、元のデータに基づいて新しいデータ・フォーマットまたはデータ・ビジュアライゼーションを作成します。

- **永久**：データは常に使用可能かつ理解可能でなければなりません。リレーショナル・データベースは、非常に検索しやすいため、データの長期保存に最適です。製造情報に関するデータベースを持っていない場合は、データベース・ソリューションを検討してください。この決定は、データ保存の方針、慣行、予算に依存します。⁵

表2のリストは、一般的なデータ・フォーマットの特性です。

表2. データ・フォーマットの比較

	バイナリ	書式なしテキスト	カンマ区切り変数(.csv)	XML(拡張可能マークアップ言語)
識別可能	特殊なツールを使用した場合のみ	少数のデータ・セットに対してのみ	適切なカラム・フォーマット・デザインが必要	重大な問題なし
検索可能	特殊なツールを使用した場合のみ	難しく、ミスのおそれあり	重大な問題なし	優れているが、XMLに関する専門知識が必要
変換可能	特殊なツールを使用した場合のみ	難しく、ミスのおそれあり	重大な問題なし	優れているが、XMLに関する専門知識が必要
永久	特殊なツールを使用した場合のみ	少数のデータ・セットに対してのみ	重大な問題なし	重大な問題なし
例： スプレッドシート解析	特殊なツールを使用した場合のみ	インポート不可	Excel、その他によってサポート	Excel 2003フォーマットの使用が可能

5 Tufte, Edward R. "The Visual Display of Quantitative Information." Graphics Press, 2001.

バイナリ・フォーマットには、自己記述ではないという根本的な問題があります。さらに、データを解釈するために別のソフトウェアが必要です。データの解釈に使用するソフトウェアによっては、変換機能が制限される場合もあります。

テキスト・ファイルは、検索および変換が難しい上に、識別も困難です。プレーン・テキスト・ファイルには正規フィールドがないため、例えば番号12のテキスト検索では、時刻の12、リミット値の12、DMM測定値の12がすべて検索される可能性があります。

カンマ区切り値(ドットcsv)テキスト・フォーマットは、Microsoft Excelにインポートしやすいので、さまざまな操作が可能です。Microsoft Excelを使用すれば、測定結果テーブルを作成して、各行に測定結果を、各列に一意の説明をつけることが簡単にできます。もう1つの利点は、ほとんどのデータ解析ソフトウェアがこのフォーマットを簡単に読み取ることができるということです。このフォーマットの欠点は、階層データを保存したり、データ・セットを簡単に解析することができないということです。列の数とタイプに関する決定を前もって行い、各列に固有のデータ・フィールド

が1つずつ含まれるようにしなければなりません。

XML⁶は自己記述で、極めて変換しやすいフォーマットであり、検索特性に優れています。XMLデータを、新しいXMLフォーマット、HTML、簡単なテキスト・フォーマットに変換するためのExtensible Stylesheet Transforms (XSLT) と呼ばれるXML言語があります。⁷ Microsoft Excel 2003などの多くのデータ解析プログラムは、XMLデータをインポートできます。⁸ 解析ツールに適合したXMLフォーマットでデータを出力できない場合は、すべてのXMLデータを必要なフォーマットに変換する比較的小さなXSLTを記述することができます。XSLTは高度な検索機能も備えているので、データ値やデータ構造の識別も非常に容易です。

製造テスト業界はすでにXMLの採用を開始しています。一部のテスト・エグゼクティブ・アプリケーションは、XMLデータでのログをサポートしています。製造テスト・データの通信用のXMLフォーマットを定義するIPC 2547⁹と呼ばれる規格も定められています。

図6は、XMLフォーマットで実行される標準的なテストの例です。“PowerTest”およびテスト・システムのハードウェア構成のテスト・リミット値の変更が可能な場合でも、テスト・シーケンスIDや別の形のテストを知りたいはずで

これが.csvファイルであったとすると、すべてのレコードに対してフィールドを作成して、これらの質問に答える必要があります。XMLを使用すれば、<TestSequence ID="32">と呼ばれるレコード・タイプを挿入し、そのレコードの中に現在のテスト・シーケンスを全部記述することができます。さらに、<TestRun>レコードの中に、そのテスト・シーケンス・レコードを参照する“IDREF”と呼ばれるXML属性を追加することも可能です。

まとめると、選択したデータ・フォーマットは、その値の時間変化に大きな影響を及ぼします。テスト・プログラムのメンテナンスのために他の人がそのデータを読み取って解釈することがどれだけ簡単または困難かを考える必要があります。

```
<?xml version="1.0" ?>
- <TestReport xmlns="urn:Agilent/EPSPG/Casper/Production">
- <Sequence name="FastTestA.tsq">
  <DUT serialnumber="0000100245" model="101" />
  <TestEnv operator="Joe1" host="fasttest3" date="1/22/04" time="01:24:31 pm GMT" />
  <Result success="0" message="PowerTest: Voltage outside Expected Range" elapsedSeconds="109" />
  <Test name="PowerTest" success="0" />
</Sequence>
- <Sequence name="FastTestA.tsq">
  <DUT serialnumber="0000100246" model="101" />
  <TestEnv operator="Joe1" host="fasttest3" date="1/22/04" time="01:29:03 pm GMT" />
  <Result success="1" elapsedSeconds="124" />
  <Test name="PowerTest" success="1" />
</Sequence>
</TestReport>
```

図6. XMLレポート・ファイル

- 6 Extensible Markup Language: <http://w3.org/xml>.
- 7 Holzner, Steve. "Inside XML." New Riders, 2000.
- 8 XML in Microsoft Office: <http://www.microsoft.com/presspass/press/2002/Oct02/10-25XMLArchitectMA.asp>.
- 9 IPC 2547: <http://webstds.ipc.org/2547/2547.htm>

ユーザ・インタフェースの設計

テスト・システムにログインする時に何が表示されるかは、ユーザ・クラスによって決まります。ユーザ・クラスとしては、オペレータ、テスト・エンジニア、サービス/校正エンジニアが考えられます。正確に記述されたSRSでは、各ユーザ・クラスが使用できるコマンドやメニューを定義しています。提供する選択肢が多いほど、混乱やミスが生じる可能性が高くなるため、各ユーザ・クラスには、ジョブの実行に必要な機能と情報だけを提供する必要があります。

安全を確保するために、各ユーザごとに一意のログインを作成し、各ユーザのログインは、適切なクラスにリンクする必要があります。

プロトタイプやストーリーボードの作成からなるUCD (User-Centered Design) と呼ばれる手法によって、GUIがユーザのニーズを満たしているかを確認することができます。一般に、テスト・システムのGUIは以下を実行できる必要があります。

1. ユーザ・クラスに基づいて動作をカスタマイズする。
2. DUTに関する詳細情報を提供または入力可能にする。
3. システムの状態に関する情報を提供する。
4. システムの状態、場合によってはその構成を制御するための操作を提供する。
5. DUTのテスト結果を表示する。

オペレータの場合、設計するインタフェースには、テスト・システムの状態(例えば、テストの実行中、休止中、停止中)が常に表示されていなければなりません。例えば、テスト・システムに実装されているライトと一緒に、PCモニタ上の色分けされた大きなグラフィカルを使用することもできます。オペレータには、DUT情報を入力するための手段はもちろん、テスト・システムの状態を制御するための手段も必要です(バーコード・スキャナによって自動的に行われる場合を除く)。

一般的には、設計するテスト・プログラムには以下が必要です。

1. テスト・シーケンスを開始/停止するためのコマンド
2. テスト結果を各種プリンタ(不具合レポートなど)に送信するためのコマンド
3. テスト・シーケンスの動作の制御(例えば、ドロップダウン・リストからのDUTの選択)

4. テスト・システムの詳細な説明を表示するための方法。テスト結果メッセージには多くの情報が含まれているので、ユーザ・エラーや多発するハードウェア上の問題のクイック診断に有用であるだけでなく、究極的にはテスト・エンジニアが工場現場を訪れる必要もなくなります。

図7に示されているユーザ・インタフェースは、量産テスト・アプリケーションでのオペレータ用に設計されたものです。オペレータは、テスト・システムにログインして、テストプランの名前とバージョンを選択し、DUT情報を入力することから始めます。ディスプレイのテスト・ステータス部分が製造テスト環境用と比べると多少目立たず分かり難いので、テスト・ステータス・ライトをテスト・システムに追加する必要があるかもしれません。

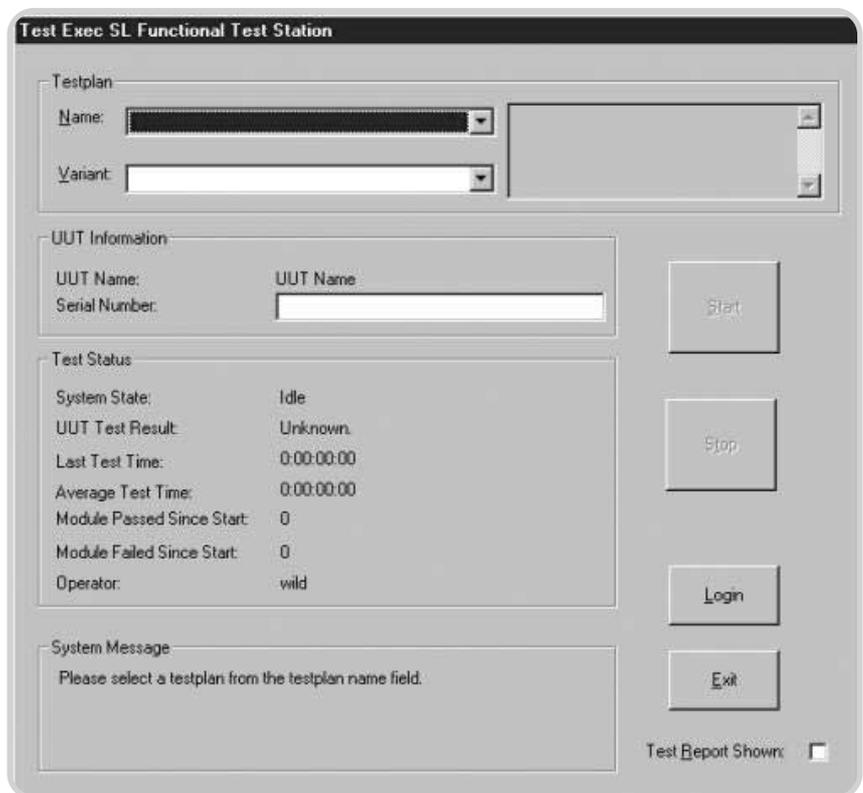


図7. 量産用ユーザ・インタフェース

9 Vredenburg, Karel, et al, "User-Centered Design, an Integrated Approach." Prentice Hall PTR, 2002.

10 Norman, Donald A., "The Design of Everyday Things." Basic Books, 2002.

システム・メッセージ・フィールドには、ユーザに対して次に何を実行すべきかの指示が出されるだけでなく、テスト結果情報も表示されます。エラー・メッセージをシステム・メッセージ・フィールドに表示して、デバッグ・プロセス中のテスト・エンジニアを支援することも可能です。

図18に示されているユーザ・インタフェースは、少量テストの状況(例えば、携帯電話の基地局)用に設計されています。このインタフェースは、テスト・シーケンスの開発やデバッグにも使用できます。このインタフェースの対象となるユーザ・クラスは、テストの目的および役割、DUT、テスト・システムの構成に関する詳細な知識を有する非常に熟練したユーザです。未熟なテスト・オペレータは、このインタフェースを有効に活用することはできません。

2種類のGUIが同じテスト・ソフトウェアを使って作成されましたが、それらは複雑さがまるで異なります。図7のオペレータ用GUIは、不必要な選択肢やソフトウェア開発者に不可欠な情報を非表示にしています。

開発環境の選択

ソフトウェア・アーキテクチャの選択の次のステップは、ソフトウェア開発環境を選択することです。選択するソフトウェア環境やツールによって、テスト・システムのコスト全体に大きな影響を与えます。ソフトウェア環境を選択する際には、ソフトウェアの購入価格だけでなくそれ以外の要素も考慮する必要があります。さらに、サポート/保守コストはもちろん、ソフトウェアの習得/使用が容易か、他の言語、デバイスまたはエンタープライズ・アプリケーションへの接続が容易かも考慮する必要があります。テスト・システムの耐用期間にわたるソフトウェアのサポート/保守コストだけでも、ハードウェア・コストを上回ってしまう可能性があります。

ソフトウェア開発環境では、C、C++、C#、VB、VB .NET、VEE、Lab VIEWなどの言語ですべてを自分で記述するものから、既製のテスト・エグゼクティブと予め記述されたサードパーティ製テストを併用するものまで、数多くの方法があります。選択し

たソフトウェア環境で、次の2つの目標を達成させる必要があります。1) 最初のテストまでの時間条件を満たす、2) テスト・スループット要件を満たす。どのくらい速くテスト・システムを立ち上げられるか? 最大スループットが得られるか?

ソフトウェア開発環境は、グラフィカルとテキストの2つに分類されます。Agilentの VEE Pro 7.0(図9を参照)やLabViewなどのグラフィカル環境は、エンジニアが習得/使用するの簡単です。その主な理由は、エンジニアが回路図を見慣れているからです。さらに、テキスト形式のプログラミング言語に比べて、小~中規模のグラフィカル・プログラムの変更は簡単です。従来、テキスト形式のプログラミング言語は製造環境で高速に動作し、高いスループットを発揮していました。今日では、グラフィカル環境とテキスト環境の動作速度の違いは小さくなっています。

グラフィカル環境の方がテキスト環境よりも使いやすいのですが、製造テスト・システムには通常はテキスト環境が用いられています。グラフィカル形式のプログラミング言語を使用しているのは、電子計測器用のコードを書いている50万人以上のユーザのうちの22%ほどです。¹¹

グラフィカル・プログラミングかテキスト・プログラミングか?

アプリケーションに適した開発環境を選択するためには、さらに詳しくそれぞれの使用モデルを理解することが重要です。

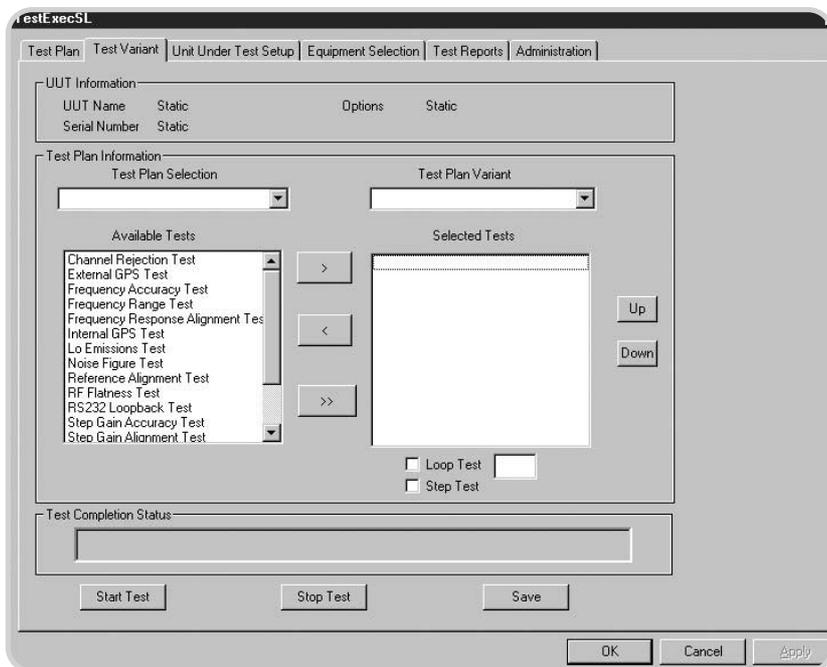


図8. ソフトウェア開発者用のインタフェース

¹¹ 開発環境の詳細については、以下を参照してください: www.agilent.co.jp/find/vee; www.softwire.com, www.ni.com/labview, and Richter, Jeffrey, Applied Microsoft .NET Framework Programming, Microsoft Press, 1 edition, January 23, 2002.

グラフィカル・プログラミングは、アイコンまたはオブジェクトと呼ばれるイメージと、これらのイメージを結ぶ線を操作することによって行われます。イメージが予め作成されたコマンドを表すのに対して、線はプログラム・フロー、制御ポイント、データの発生/消費の方法を表します。アイコンと接続線は、統合型開発環境(例えば、ソフトウェア・プログラム)に組み込まれています。

多くのグラフィカル・プログラミング環境が、プログラミング環境の起動を必要としないコンパイル済みプログラムまたはパッケージ済みプログラムの作成機能を備えています。電子計測エンジニアを対象とするグラフィカル・プログラミング環境はいくつかあります。これらのプログラムは、幅広いI/Oと機器ドライバ、電子計測固有の演算機能、グラフィカル処理機能を備えています。

グラフィカル・プログラミング言語がテキスト形式のプログラミング言語より優れている点を以下に示します。

1. **複雑な文法がない**：線で結ばれたアイコンの集まりで表されるプログラム命令の方が速く理解できます。
2. **同時動作も容易**：複数の同時動作は、データフロー・モデルと呼ばれるものに依存しますが、実行前にすべてのデータをコマンドによって使用可能にする必要があります。これは、C++やJavaなどのテキスト形式のプログラミング言語のマルチスレッドを使うよりも簡単です。
3. **実在するものの象徴を使用できる**：コマンドを表すアイコンは、アイコンによって実行される動作に相当する実在のものを表す象徴(イメージ)です。ほとんどのテスト・エンジニアは、テキスト・プログラミングよりもグラフィカル・プログラミングの方が直観的で、使いやすいと思っているはずで

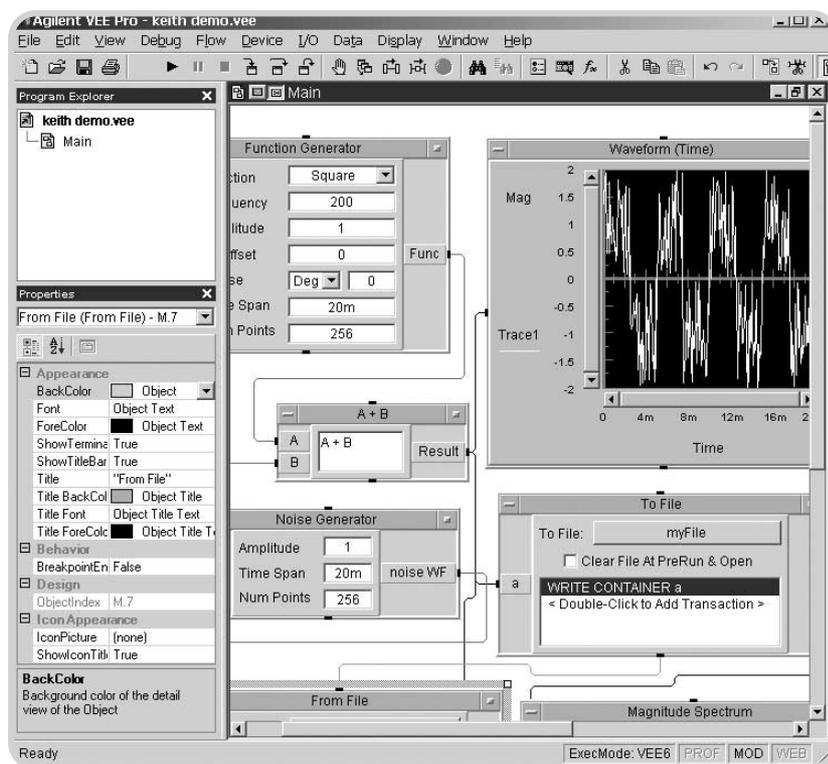


図9. Agilent VEE Pro 7.0のグラフィカル・プログラミング環境

Agilent VEE Pro 7.0と T&M Programmers Toolkit

Agilent VEE Pro 7.0

- 概要：使いやすく、強力なグラフィカル・プログラミング環境
- アプリケーション：データ収集、デザイン、少量製造テスト
- 目的：機器データを収集/解析するためのグラフィカル・プログラムの作成
- 機能：簡単なテスト・システムの制御、シーケンス処理、Microsoft .NET Frameworkのサポート、MATLAB® 解析/視覚化、ActiveXのフル・サポート

Agilent T&M Programmers Toolkit

- 概要：Visual Studio .NETに統合されたテスト・コード開発(VB .NET、C++またはC#で)
- アプリケーション：デザインの特性評価、デザインの検証、製造
- 目的：PCの標準環境での様々なドライバを使った複雑なプログラムの記述
- 機能：測定器のI/Oおよび通信、テスト・コードのデバッグ、データ収集、表示/解析、IVI-C、IVI-COM、VXIplug&playドライバのサポート

4. **プロトタイプ的高速作成**：グラフィカル・プログラミング言語は、システムのプロトタイプをすばやく作成できます。プロトタイプの作成はどのような規模のシステムの開発にも役立ちます。¹²
5. **既存のプログラムの共有／習得が簡単**：実在のモデルを視覚的に使用することによって、既存のプログラムの共有／習得および生産性の向上を容易に実現できます。¹³

テキスト形式のプログラミング言語は、特殊な用語と文法を使ってプログラムの動作や流れを表します。すべてのテキスト形式のプログラミング言語とは限りませんが、ほとんどのテキスト形式のプログラミング言語がオープン・スタンダードに準拠しています。したがって、プログラミング環境やソフトウェア・ツールについては、ユーザがベンダを選択することができます。また、グラフィカル言語よりも広く用いられているので、はるかに多くのサードパーティ製ドライバ、ツール、アドインを利用できるという利点があります。

テキスト形式のプログラミング言語がグラフィカル言語よりも優れている点を以下に示します。

1. テキスト形式のプログラミング言語は、大規模で包括的なプログラムの作成に適しています。

2. 大規模プログラムの場合、テキスト形式のプログラミング言語は、検索したり理解するのが簡単です。情報が複雑または小さ過ぎて見にくくなるので、人間は一度に約50個のグラフィカル・オブジェクトしか見ることはできません。¹⁴ プログラム内をあちこち移動してオブジェクトをすべて見るしかない場合は、ユーザは制御ラインやデータ・ラインを見失ってしまい、プログラムの全体的な流れを理解することが難しいと感じる可能性があります。しかし、プログラムの大きな動作をより小さな動作に分割することにより、大きなグラフィカル・プログラムをより分かりやすくすることができます。これは、機能分割と呼ばれ、連続したコマンドを「ブラック・ボックス」に入れることによって実現します。次に、コマンドを機能ブロックに送り、必要に応じてその出力を受け取ります。グラフィカル環境で大規模プログラムを作成する場合、グラフィカル・プログラミング言語がこの機能分割¹⁵をサポートしていることを確かめてください。

3. テキスト形式のプログラミング言語を用いることによってシステム全体のスループットを高めることはできませんが、測定器の動作に費やされる時間ももっと重大な影響をもたらします。例えば、どのプログラミング言語が用いられているか(グラフィカルかテキストか)に関係なく、DUTから測定器へのスイッチングの効率が悪いテスト・システムでは、性能にマイナスの影響を与えます。

4. テキスト形式のプログラミング言語に関しては、開発環境も豊富です。例えば、開発環境を備えたグラフィカル・プログラミング言語が複数のベンダから提供されることはほとんどありません。このため、今日のグラフィカル言語には、ベンダ間の競争によるメリットはほとんどありません。

テキスト・プログラミングは、よりオープンになり、グラフィカル・プログラミングは、より習得／理解が簡単になっています。表3は、2種類のプログラミング環境の違いをまとめたものです。

オープン・スタンダードへの取組み

グラフィカル・プログラミングまたはテキスト・プログラミングのどちらかを選択する以外にも、選択する環境が業界標準に基づいているか、ベンダ独自の技術に基づいているかを考慮する必要があります。C++、Visual Basic、C#はすべて、業界標準のプログラミング環境です。Agilent VEE Pro 7.0は.NETなどの業界標準のテクノロジーへのアクセスは可能ですが、Agilent VEE ProおよびNI LabVIEWは独自の開発環境です。

業界標準と独自の開発環境のどちらかを選択する際に考慮すべきいくつかの要因として、1) コスト、2) 業界のサポート、3) アップグレーダビリティ、4) 拡張性があります。

表3. グラフィカル・プログラミングとテキスト・プログラミング

	グラフィカル	テキスト
自由／オープン	オープン・スタンダードがほとんどない、拡張性に劣る	オープン・スタンダードが多数を占める、非常に拡張性が高い
迅速なプロトタイプの作成	優れたテスト／測定用プロトタイプ作成機能	いくつかのコード・ウィザード(例えば、T&M Programmers Toolkit)が用意されているが、低速
テスト／測定のサポート	テスト／測定を対象とした数種類のグラフィカル環境、多数のドライバ	数種類のテスト／測定用のサードパーティ製ツールが用意されている、多数のドライバ
サードパーティ製ツール	数百	数千
習得／共有可能	簡単にプログラムを取り出して使用できる プログラムは共有しやすい	小さいか、非常によくデザインされている場合のみ

12 Rahman, Jamal and Lothar, Wenzel, "The Applicability of Visual Programming to Large Real-World Applications," 1995, <http://www.computer.org/conferences/vi95/html-papers/wenzel/paper.html>.

13 Blackwell, Alan F. and Green, T.R.G., "Does Metaphor Increase Visual Language Usability?," IEEE Symposium on Visual Languages VL'99, 1999, pp. 246-253.

14 Begel, Andrew, "LogoBlocks: A Graphical Programming Language for Interacting with the World," 1996, <http://www.cs.berkeley.edu/~abegel/mit/begel-aup.pdf>.

15 Glinert, E. P., "Visual Programming Environments," IEEE Computer Society Press, 1990.

オープン・スタンダードのプログラミング開発環境は、それらに対応する独自の開発環境より優れた機能セットを備えていて、しかも安価です。すなわち、オープン・スタンダードの環境はより熾烈な競争を生み出し、熾烈な競争は価格を下げ、革新をもたらす傾向にあります。

オープン・スタンダードの言語は、ソフトウェア・ツール・ベンダとオープン・ソース開発者の両者が大きな関心を持っています。これらのグループはどちらも、テスト・システムのプログラムのニーズを理解することに相当な時間を費やし、結果として、それらのニーズを満たすために無料と有料の両方のツールやアプリケーションを開発しています。1つのよい例が、市場に出回っている、ベンダとエンドユーザの両者による大量のCおよびC++ライブラリです。開発者がゼロから作成するよりも、ソフトウェア(数学解析ライブラリなど)を購入する方が速く、安いので、これらのライブラリによって開発に要する時間とコストが削減されます。

ほとんどの独自の環境は新しい技術をすぐに活かすことができないので、市場投入までの時間の面でもオープン・スタンダードの環境は優位です。新しいプログラミング技術は、最も一般的なオープン・スタンダードのプログラミング言語のことを考えて開発されています。新しい技術を活用した新しいバージョンの専用ソフトウェアをベンダがリリースするにはもっと時間がかかります。

.NET Framework : .NET Frameworkは、コンピュータのプログラミング用のオープン、マルチプラットフォーム、マルチベンダのソフトウェア・テクノロジーです。C#言語は、標準化団体にオープン言語として提出されました。オープン・スタンダードの基礎となっている.NET「共通言語インフラストラクチャ」テクノロジーは、Microsoft社のWindowsやLinuxなどの複数のオペレーティング・システムで使用できます。

.NETテクノロジーは、Web開発とPCソフトウェア開発の両方の環境に適用でき、最高のサポートも得られます。.NETテクノロジーには、Javaの利点が多く見られますが、Javaの欠点の多くは排除されています。例えば、.NETテクノロジーは、プログラマのメモリ・リークをなくし、ソフトウェアの配備を容易にすると共に、システムやGUIの開発用の質の高いアプリケーション・プログラミング・インタフェース(API)を提供しています。.NETテクノロジーは、ジャストインタイム(JIT)コンパイラによって完全にコンパイルされます。JITコンパイラは、オペレーティング・システム(OS)/プラットフォーム依存のコードを解釈して、ターゲット・プラットフォーム用のマシン・レベルのコードを作成します。

実行時に.NET Frameworkをロードするのに多少のオーバヘッドが必要ですが、.NETで書かれたプログラムには互換性があり、C/C++で書かれたプログラムよりも高速に実行されます。¹⁶ プログラムが.NET環境でより高速に実行できる理由は、従来の言語ではリンカの動作に固有の効率の悪さがあるからです。¹⁷

プログラマを対象とした調査や数多くのケース・スタディでは、.NET環境によって、プログラマの従来の環境よりも生産性が大幅に向上していることが明らかになっています。¹⁸

.NET Framework(.NET言語によって使用されるAPIサービスやヘルパ・コードの集まり)は、Visual Studio .NETと同じものではありません。Visual Studio .NETは、.NET技術をサポートするMicrosoft社の開発プログラミング環境です。図10に示されているように、さまざまなベンダから複数の.NET開発環境やプログラミング言語が発売されており、複数のプラットフォームでサポートされています。

最もよく知られている.NET言語としては、C#とVisual Basic(VB).NETがあります。C#は、構造や機能の面ではJavaと非常に似ていますが、文法はC++を進化させたものです。オブジェクト指向や例外処理に詳しいC++プログラマなら、C#プログラミング環境に簡単に移行できます。

16 Wilson, Matthew, "Does C# Measure Up?," Windows Developer, Volume 2, Issue 13, Fall 2003, <http://www.wd-mag.com/wdn/webextra/2003/0313>

17 Johnson, Mark S. and Miller Terrence C., "Effectiveness of a machine-level, global optimizer," 1986, <http://portal.acm.org/citation.cfm?id=13321&dl=ACM&coll=portal>

18 <http://www.microsoft.com/net/cases-studies>

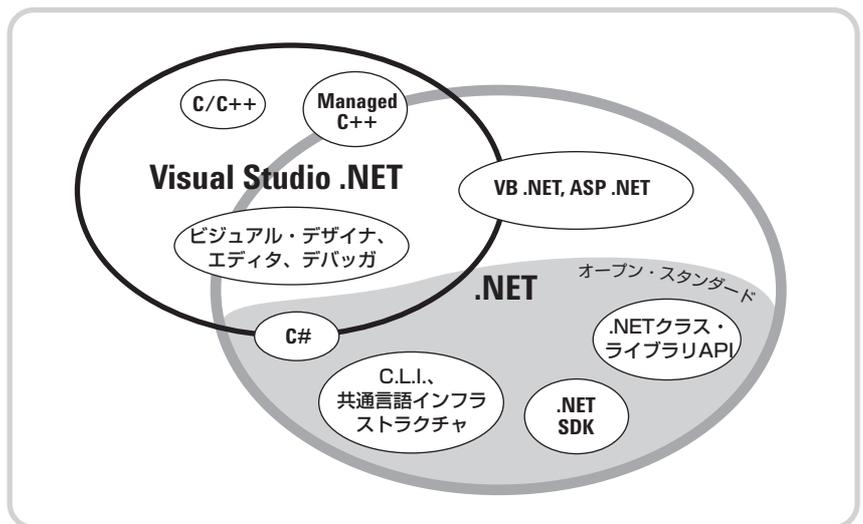


図10. .NET Frameworkのプログラミング言語

VB .NETは、Visual Basic 6からのアップグレードです。既存のVB 6アプリケーションを使用しているエンジニアは、アップグレード・ウィザードを使ってVB .NETに移行する必要があります。アップグレード・プロセスが完了したら、.NETアプリケーションにアクセスして、.NETが提供する追加機能と柔軟性が得られます。

Microsoft社のC++言語も、Managed C++と呼ばれる新しいバージョンを含むように強化されています。Managed C++は、.NETソフトウェア内でのコールの実行を簡単にします。Microsoft社は、オリジナルのUnmanaged C++をVisual Studio .NETでも提供しています。

従来のプログラミング技術よりも.NETの方が著しく優れている点の1つが、拡張性です。Microsoft社は、Windowsプログラマが経験していたDLLの実装時のストレスをなくすために、.NETを設計しました。すでに数多くの.NET用のサードパーティ製ツールが出回っています。これらのサードパーティ製の制御機能の多く(例えば、高度なグラフ用ビジュアル制御機能)は、テスト・システムのプログラマにとって便利です。さらに、Agilent Technologies、National Instruments社、Measurement Computing社などの数社の電子計測器メーカーは、.NET対応ツールを発売しました。発売されている.NET対応ツールの一覧については、Microsoft社の.NETパートナーのWebサイト(www.vsipartners.com)をご覧ください。

Agilent Technologies初のVisual Studio .NET用アドインは、Programmers Toolkitと呼ばれます(このアプリケーション・ノートの11ページの右側に掲載されている補足説明を参照してください)。T&M Programmers Toolkitは、I/Oツール、グラフ作成/数学ライブラリ、T&M専用ヘルプ、サンプル・ジェネレータ、機器ドライバなどのソフトウェア用の.NETラッパーを備えています。T&M Programmers Toolkitは、Visual Studio環境に完全に統合されています。Agilent Technologiesのソリューションの詳細については、<http://www.agilent.co.jp/find/toolkit>または<http://www.agilent.co.jp/find/iolib>をご覧ください。.NET関連のI/Oソース・ファイル(Agilent I/Oライブラリでも動作)をダウンロードするには、アジレント・ディベロッパ・ネットワーク(ADN)のWebサイト<http://www.agilent.co.jp/find/adn>をご覧ください。

テスト・シーケンスの開発

2,500以上の電子計測器ユーザを対象にした調査では、回答者の93%が、複数のテスト機器を使用している、またテスト機器をPCに接続していると答えています。そのうちの37%は、市販のテスト・エグゼクティブをテスト・シーケンスに用いています。これらの回答者のうちの残りの56%は社内すなわち「自社開発」のソフトウェアをテスト・シーケンスに使用しています。

テスト・エグゼクティブは、テストを既定の順番で実行するように設計されたソフトウェアです。テスト・シーケンス内のテストのいずれかに不合格になると、DUTは不合格になります。テスト・エグゼクティブは、長年にわたって、柔軟性と機能の両面で大幅な向上が図られて来ました。第1世代のテスト・エグゼクティブは、言語固有のもので、基幹の製造環境に使えるほど高性能ではありませんでした。Agilent TechnologiesのTxSLやNIのTestStandなどの第2世代のテスト・エグゼクティブは、性能は向上していますが、価格も高くなっています。またこれらのテスト・エグゼクティブは、少量生産に求められる柔軟性に欠けています。

テスト・シーケンス内の各テストは、個別のモジュールです。市販のテスト・エグゼクティブには標準的なテスト・モジュールが付属していますが、ユーザが既存のテスト・モジュールをカスタマイズすることももちろん、ゼロから追加のテスト・モジュールを作成することも可能です。テスト・エグゼクティブは、テスト・モジュールとの間でやり取りするデータを制御し、すべてのデータを収集/解析した後で、DUTの合否を判定します。

テスト・エグゼクティブを使用する理由の1つは、テスト・システムを作成するための構造化された枠組みが提供されるためです。テスト・エグゼクティブは、中程度から大量生産テスト環境で最も威力を発揮します。

テスト・エグゼクティブは、シーケンスのデザイン、個々のテストのデザイン、テストのリミット、構成の管理が別々のタスクとして扱われるように書かれています。これら3つのタスクを別々にすることで、柔軟性が高まり、品質が向上し、コードの再利用の機会が増えます。個々のタスクをそれぞれつなげて完全なプログラムにするのに必要なインフラやヘルパ・サービスを提供するのが、テスト・エグゼクティブです。

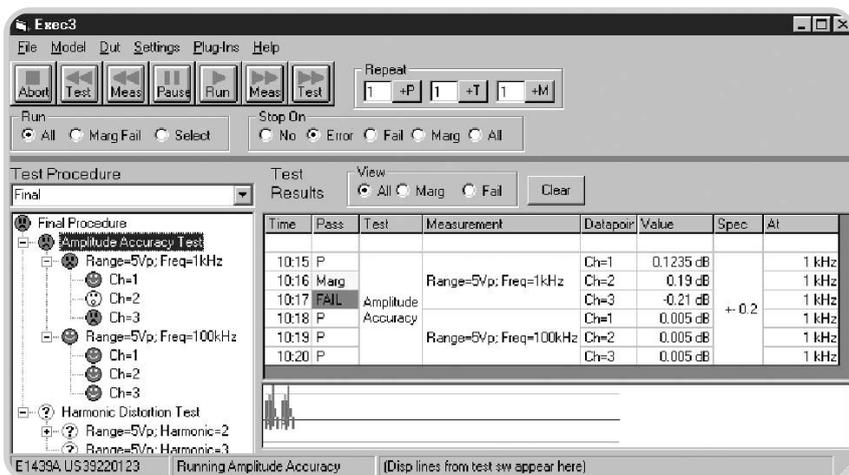


図 11. テスト・エグゼクティブのテスト・シーケンス

テスト・エグゼクティブの最も重要な機能の1つが、テスト・シーケンサです。図11の左側に示されているように、テスト・シーケンサは、デザイン・モードで操作できるテスト・シーケンスです。「テスト・ルーピング」などのこのシーケンスの柔軟性のレベルは、テスト・エグゼクティブによって異なります。

最低でも、テスト・エグゼクティブは以下のタスクを実行できる必要があります。

1. 独自のデータ収集モデルを使って、結果（および追加データ）を捕捉する。
2. テストのリミット値およびテスト・セットアップのデータをトラッキングし、実行時にセットアップ・データをテストに渡す。
3. リミット・チェックを提供する。
4. テスト結果（合否レポート）のランタイム解析を提供する。

さらに、テスト・エグゼクティブには、テスト・モジュールを保持するため（およびテストの再利用を促進するため）に、ソフトウェア・リポジトリが含まれている場合もあります。ソフトウェア・リポジトリにより、テスト・エンジニアは、テスト・モジュールのリポジトリ内の検索を行って、特定のテストを探ることができます。社内の全エンジニアが同意して1つのテスト・エグゼクティブに決定すれば、異なる製品／製造グループ間でテスト・モジュールを共有できます。

テスト・エグゼクティブは、テスト・システムの制御／データ・ラインのレイアウト（およびすべてのスイッチ・ボックス）をDUTや測定器のI/Oピンにマッピング可能なスイッチング・モデルを使用することもできます。このため、テスト・エンジニアは、システムの配線方法に神経を使うのではなく、測定器とDUTの間の論理的な接続に集中できます。

最後に、テスト・エグゼクティブの中には、オペレータ・インタフェースを構築するためのツールを備えているものもあります。この機能は、以前検討した開発環境の1つを使用するよりも柔軟性は劣っていますが、高速かつ簡単な方法を提供します。

ソフトウェアの再利用

標準ライブラリやオペレーティング・システムのAPIは別として、ほとんどのソフトウェアの再利用が楽観的になりがちです。典型的な再利用のシナリオは、プログラマーが問題に直面した時に、仕事仲間が同様の問題に対処したこと思い出した場合です。プログラマーは、これまでのプログラムの古いソース・コードをくまなく検索して、必要なコードを見つけ出します。コードが見つかった場合は、プログラマーは、ソフトウェアを現在のテスト状況に適用できるのか、どうすれば適用できるのかを決定します。変更を行った後、ソフトウェアを再検証する必要があります。ほとんどの場合、ソフトウェアをゼロから作成するよりも時間がかかりません。

上述のシナリオは、システムティックなソフトウェアの再利用方法を用いれば改善したかもしれません。システムティックな方法の利点は、新しいテスト状況に適合するテスト・コードを検索し、見つけ出し、検証し、採用するのに要する時間が短くなるという点にあります。システムティックな再利用方法は、標準化された会社の方針や慣行の順守はもちろん、特殊なコーディング方式や様式への準拠が必要です。

本書では、全社を挙げてのシステムティックな再利用プログラムを実施する上で注意すべき事柄すべてを検討することはしませんが、チームのため、延いては会社のために、システムティックな方法を実現できるようにするために、あなたが下すことができる決定はいくつかあります。再利用についての検討は、システム要件を収集してからソフトウェアの開発を始めるまでの間に着手する必要があります。

専用のテキスト・エグゼクティブかカスタム・ソフトウェアか？

独自のテスト・エグゼクティブを作成すべきか、既製品を購入すべきかをどのように決定しますか？ 考慮すべきいくつかの要因を以下に示します。

1. まず最初に、テスト・エグゼクティブが必要かどうかということです。比較的固定されたテスト・シーケンスがない場合は、テスト・エグゼクティブは適していません。
2. 社内に自社開発テスト・エグゼクティブが存在する、あるいはいくつもの自社開発テスト・エグゼクティブが存在する場合は、それらの品質、機能、サポートの有無、テスト・コレクション、またはそれらに使用可能なその他の補助ソフトウェアを詳細に調べる必要があります。
3. 妥当な選択肢が見つかったら、既存のコードを移植して既製のテスト・エグゼクティブを使用するためのコストを考慮します。
4. サポート品質や機能から、既製のテキスト・エグゼクティブを使用する場合もあります。
5. 既製のテスト・エグゼクティブの方が外部委託を行う場合にも優れた点があります。サードパーティのソフトウェア請負業者やコンサルタントは、すでにこうしたテスト・エグゼクティブに関する知識を持っている可能性があるため、サードパーティ製ライブラリを使用できる可能性があります。
6. 既製のテスト・エグゼクティブには、マニュアルが一式含まれているはずで

既製のテスト・エグゼクティブを用いることにした場合は、サービスやサポートの品質についても検討します。

デザインの再利用プロセス

デザインの再利用プロセスの最初のステップは、領域解析をすることです。これは、1) ソフトウェアの機能およびパーツを体系的に解析し、2) この情報を用いてコンポーネント・タイプとアルゴリズムのソフトウェア・アーキテクチャを構築することにより実行します。

次に、ソフトウェアの自然境界を探します。自然境界を見つけ出して文書化するというソフトウェア設計手順の1つに、デザイン・パターンと呼ばれるものがあります。¹⁹ 自然境界を見つけるには、1つのタイプの動作またはデータ・セットが別のタイプの動作またはデータ・セットとリンクする領域に注目します。これらの領域を、別々のモジュールにグループ分けし、それに従って文書化することができます。文書化が済んだら、同じタイプのモジュールを互いに交換することができます。

モジュール、コンポーネント、個々のパーツを識別し、収集し、文書化したら、リポジトリに入れたり、仕事仲間に渡す前に、それらをテストする必要があります。テストにより、プロセスの後半で問題に直面することがなくなります。

最後に、再利用可能なコンポーネントは、それらの存在を知っている場合にのみ再利用可能です。どんなコンポーネントか、何を実行するのかに基づいて、誰もが検索できるモジュール用のリポジトリ (例えば、リレーショナル・データベース) が必要です。

デザインの再利用の例

個々のテスト・モジュールのデザインの再利用の1つのようなモデルが、テスト・エグゼクティブです。理由を以下に示します。

1. テスト・エグゼクティブには、テスト・ソフトウェアを、交換可能なテスト・シーケンス、リミット・チェック、テスト・シーケンス、テスト・リミット値データに分割するものがあります。²⁰
2. テスト・エグゼクティブは、モジュールの概念に依存します。例えば、シーケンスのデータ・タイプ、シーケンス実行処理、テスト・タイプを含めて、単一の合否判定を実行する機能を備えたモジュールを使用することができます。
3. テスト・エグゼクティブを用いれば、テスト・コードを変更しなくても、さまざまなテスト・シーケンスのテストを再利用することができます。シーケンスは、現在のテスト・シーケンスに合わせて処理をカスタマイズするために必要なデータを提供します。
4. テスト・エグゼクティブは、テスト・シーケンスまたはテスト・エグゼクティブ・アプリケーションからテストを個別のモジュールまたはファイルに入れて保持します。これにより、再コンパイルしなくても簡単に、テストをスワップ・イン/スワップ・アウトすることができます。
5. テスト・エグゼクティブには、ユーザー独自のカスタム・リミット・チェックやシーケンスの記述が可能なものがあります。

これらのモジュールはどれも相互運用することができます。これは、テスト・エグゼクティブが明確なアプリケーション・プログラミング・インタフェース (API) を各モジュールに使用するためです。これらのモジュールは、テスト・エグゼクティブ内のさまざまなタイプのデータや機能の自然境界上に置かれます。

ソフトウェアの機能、タイプ、データの自然境界に依存するアーキテクチャを持つコードについても、同様に再利用することができます。これを実現するためには、テストのリミット値などの頻繁に変更される情報をデータ・ファイルに入れます。テスト・クラスなどの柔軟性の低いエレメントをタイプまたは「クラス」に入れます。ファンクションまたは「プロシージャ」は、柔軟性が最も低いエレメントとして確保しておきます。

デザインの再利用と.NET

ソフトウェアの境界の定義は、プログラミング言語やソフトウェア環境によって影響を受けるわけではありませんが、ソフトウェアのモジュラ構造を維持し、交換可能な状態に保つという点で他より優れている環境もあります。

.NETは、事業部内や会社内でのソフトウェア再利用プログラムの立案を容易にするソフトウェア・ツールを提供しています。.NETはオブジェクト指向であるため、テストやシーケンスなどの様々なオブジェクト間の境界を表すのに最適です。Cなどの非オブジェクト・ベースの言語の場合は、コンテキスト依存ヘルプやコンパイル時エラー・チェックがなくても、どの機能がどのオブジェクトに適用されるかトラッキングすることができます。

.NETには、高度なバージョン機能や配備機能が組み込まれています。さらに、.NETには、特定のバージョンの外部ライブラリだけを受け入れることをWindowsに知らせる機能もあります。これによって、外部ライブラリ (DLL) への依存というこれまでのバージョンのWindowsに対する不満の1つは解消されますが、その一方でDLLが変更されると、ソフトウェアは正しく機能しなくなります。

19 Shalloway, Alan and Trott, James R., "Design Patterns Explained: A New Perspective on Object-Oriented Design," Addison-Wesley Pub Co, 2001.

20 これは、電子計測の分野に固有のデザイン・パターンの1つのような例です。

デザインの再利用のメリット

まとめると、デザインを再利用する理由は、ソフトウェア品質の向上、ソフトウェア開発の効率の向上、専門知識の有効活用です。

デザインの再利用は、いくつかの理由で品質を高めます。まず、追加のアーキテクチャ解析の結果、ソフトウェア・エラーが減少し、システムのデザイン、柔軟性、透明性が向上します。適切な再利用により、徹底的にテスト/検証されたコンポーネントを利用できるので、新たな不具合が生じる可能性が減少します。

デザインの再利用は、重複した手間を省くことにより、ソフトウェア開発の効率を高めます。コンポーネントの設計、実装、テストは一度だけで済みます。適切な再利用により、新しいコンポーネントをリライトや再作成するのではなく、既存のコンポーネントの再利用を容易にします。

デザインの再利用により、組織の専門知識が活かされます。例えば、ほとんどのソフトウェア開発者は、特定の技能の専門化に時間を費やし、それらの技能に基づいてコンポーネントを記述します。時間が経つにつれて、再利用可能なコンポーネントは、組織内で最もよく知っている知識となります。会社の専門技術や深い知識は、再利用可能な豊富なコンポーネントから明らかになります。

これらのメリットは理論的ではありません。航空宇宙局 (NASA) Goddard宇宙飛行センターにあるソフトウェア技術研究所では、航空力学部門でのソフトウェア製品の開発にソフトウェアの再利用を導入することにより、大きな効果を上げました。ソフトウェア技術研究所によると、NASAでは、1行のコードを記述するのに要する労力の35%の削減、1日当たりの生産性の53%の向上、コード品質の87%の向上を実現しています。²¹

デザインの再利用についてのまとめ

会社全体でシステムティックなデザインの再利用を実現するためには、経営陣が再利用に必要な余分な作業を認める必要があります。投資に失敗してやるべきことをちゃんと成し遂げることができないと、そのうちにストレスが生じ、時間を無駄にすることになります。必要とするすべてのエンジニアが、1つ以上のソフトウェア・コンポーネント・リポジトリを利用できるようにする必要があります。また、再利用するつもりコードの著作権や特許制限に注意する必要があります。例えば、別の企業との契約の下でソフトウェアが書かれている場合には、その企業がそのコードの独占権を持っている場合があります。²²

まとめ

テスト・システム用のコードを記述する前に、システムのソフトウェア・アーキテクチャについて、多くのことを決めておく必要があります。システムに何を行わせたいのか、どのように動作させるべきかを定義する詳細なソフトウェア要件仕様書を作成することから始めてください。SRSには、エンド・ユーザとシステムの対話方法はもちろん、データの収集、保存、解析、表示方法の概要を盛り込む必要があります。

前もって行う必要のあるもう1つの重要な決定に、コードの記述に使用するプログラミング環境と言語があります。Visual Studio .NETなどの標準環境を使用すれば、柔軟性が最大限に高まるだけでなく、ソフトウェアの耐用年数を延ばすことができます。Microsoft社のVisual Studio .NETとAgilent TechnologiesのT&M Programmers Toolkitを併用することにより、Agilent VEE Pro 7.0などの様々な言語で書かれたオブジェクトをラッピングすることができます。これにより、それらのオブジェクトを新しいプログラミング環境に転送して、既存システムのコードへの投資を最大限に活用することができます。

グラフィカル環境とテキスト環境のいずれを選択するかは、システムの規模や複雑さ、ユーザの技能、社内規格、プログラミング・チームの規模によって決まります。決定には、通常、生産性が高まるのはどちらの環境か(グラフィカルかテキストか)ということも考慮します。テキスト環境は、最高の能力と柔軟性をもたらすと共に、高スループットを実現するので、ほとんどの場合、大規模な高スループット製造テスト・システム用のコードの作成に最適です。

最後に、既製品のテスト・エグゼクティブを使用するか、独自のテスト・ルーチンを記述するかを決定しなければなりません。テスト・エグゼクティブは、テスト・システムの開発を加速し、コストの削減を実現しますが、トレーニングへの先行投資が必要です。2、3のテストしか行わない場合には、独自のコードを記述する方が適している可能性があります。

21 Proceedings of the Sixteenth Annual NASA/Goddard Software Engineering Workshop: Experiments in Software Engineering Technology, Software Engineering Laboratory, December 1991.

22 Defter, Frank W, et al, "Software Reuse: Major Issues Need to Be Resolved Before Benefits Can Be Achieved," United States General Accounting Office, 1993, <http://www.defenselink.mil/nii/bpr/bprcd/vol2/272c.pdf>.

付録

用語集

ActiveX：COM対応のソフトウェア・モジュールをカプセル化して、標準的なPCアプリケーションで使用できるようにするための標準的な方法。ActiveXコントロールは、作成された環境に関係なく、ActiveX対応のアプリケーションで使用することができます。

ADE (アプリケーション開発環境)：統合型のソフトウェア開発プログラム・スイート。ADEには、テキスト・エディタ、コンパイラ、デバッガ、さらにアプリケーション・プログラムの作成、保守、デバッグに用いられるその他のツールが含まれている場合があります。例：Microsoft Visual Studio

API (アプリケーション・プログラミング・インタフェース)：APIは明確に定義されたソフトウェア・ルーチンで、これによりアプリケーション・プログラムはオペレーティング・システムやライブラリによって提供される機能やサービスにアクセスすることができます。例：IVIドライバ

C# (「Cシャープ」と発音)：C/C++の複雑さを取り除いたCに似た新しいコンポーネント指向言語。

COM：Microsoft COMを参照。

Direct I/O：測定器に直接送られるコマンド。ドライバによる利点も、ドライバによる干渉もありません。

SCPIの例：SENSe:VOLTage:RANGe:AUTO

ドライバ (またはデバイス・ドライバ)：コンピュータに常駐している機能の集まりで、周辺機器の制御に用いられます。

DLL (Dynamic Link Library：ダイナミック・リンク・ライブラリ)：アプリケーション・プログラムと結びつけられている実行可能プログラムまたはデータ・ファイルで、必要な時だけロードされるため、メモリの使用量が減少します。DLL内の機能やデータは、複数のアプリケーションで同時に共有することができます。

IDE (統合型開発環境)：ADEを参照。

入出力(I/O) レイヤ：周辺機器からデータを収集し、周辺機器にコマンドを発行するソフトウェア。VISA関数ライブラリは、アプリケーション・プログラムやドライバによる周辺機器へのアクセスを可能にするI/Oレイヤの例です。

IVI (交換可能仮想測定器)：IVI Foundation (<http://www.ivifoundation.org>) によって定義された標準測定器ドライバ・モデルで、エンジニアはコードを書き直さなくてもメーカーの異なる測定器に交換することができます。

IVI COMドライバ (IVI Componentドライバとも呼ばれる)：IVI COMは、IVIドライバをCOMオブジェクトとして表します。IVI COMは洗練された方法で処理を行ない、簡単かつ一貫した方法でコマンドを測定器に送信するため、開発環境のインテリジェンス機能とメリットをフル活用することができます。複数の測定器を使用している場合でも同じです。

ライブラリ：再利用可能なソフトウェアの処理や機能が含まれているファイルで、他のプログラムが使用することを意図しています。Cベースのライブラリ、Visual Basicライブラリ、.NETライブラリ、COMライブラリなどのソフトウェア技術をベースとしたものがあり得ます。

Microsoft COM (コンポーネント・オブジェクト・モデル)：ソフトウェア・コンポーネントの概念は、コンポーネントが同じインタフェースを表し、同じ機能を実行し、互換性がある限り、ハードウェア・コンポーネントの概念と似ています。ソフトウェア・コンポーネントは、DLLの拡張です。

Microsoft社は、アプリケーションを再構築しなくても、既存のアプリケーション・プログラムで使用できるソフトウェア・コンポーネントをソフトウェア・メーカーが作成できるようにするために、COM規格を開発しました。この機能より、電子計測器やそれらのCOMベースのIVI-Componentドライバの交換が可能です。

.NET Framework：.NET Frameworkは、Windows環境でのアプリケーション開発を容易にするオブジェクト指向のAPIです。.NET Frameworkは主に、共通言語ランタイムと.NET Frameworkクラス・ライブラリの2つのコンポーネントによって構成されています。誰でも新しいフレームワークを追加することができます。

Plug&Playドライバ (汎用機器ドライバとも呼ばれる)：専用ドライバの重要なカテゴリ。Plug&Playドライバ規格は、本来、VXI機器用に開発されたもので、VXIPlug&Play規格として知られていました。これらの規格は、VIX以外の機器に適用された時に、単に“Plug&Play”ドライバとして知られるようになりました。ライブラリ関数はアクセス可能なC言語ソースなので、C、Basic、VEE、LabVIEW、LabWindows/CVIで書かれたプログラムから呼び出すことができます。

SCPI (Standard Commands for Programmable Instrumentation)：SCPIは、計測システム内のプログラム可能な電子計測器を制御するための標準コマンドを定義しています。詳細については、<http://www.scpiconsortium.org>をご覧ください。例については、“Direct I/O”を参照してください。

汎用ドライバ：Plug&Playドライバの別名。

VISA (Virtual Instrument Software Architecture)：VISA規格は、VXIPlug&PlayFoundationによって標準化されました。VXIPlug&Play規格に準拠しているドライバは、常にVISAライブラリ経由でI/Oを実行します。Plug&Playドライバを使用している場合は、VISA I/Oライブラリが必要です。VISA規格は、物理インタフェースを経由する場合と同様の一般的な関数呼出しを提供することを目的としていました。実際には、VISAライブラリは、ベンダのインタフェースに固有のものです。

VISA-COM : VISA-COMライブラリは、VISA仕様のコンピとして開発されたI/O用のCOMインタフェースです。VISA-COM I/Oは、COMベースのAPIでVISAのサービスを提供します。VISA-COMにはVISAでは提供されていない上位レベルのサービスがいくつか含まれていますが、下位レベルのI/O通信機能に関しては、VISA-COMはVISAのサブセットです。Agilent VISA-COMは、IVI Componentドライバによって使用されるため、Agilent VISAもインストールする必要があります。

VXIplug&play : メーカーの異なるVXI機器間の相互運用性を実現するためのハードウェア/ソフトウェア標準。詳細については、<http://www.vxipnp.org>をご覧ください。

XML (拡張可能マークアップ言語) : 構造化されたデータの交換用テキスト・マークアップ言語を構成するSGMLのサブセット。ユニコード規格は、XMLコンテンツ用の参照文字セットです。

Related literature

データシート

Agilent VEE Pro, 7.0
カタログ番号5988-6302JA

Agilent Toolkit,
www.agilent.co.jp/find/toolkit

アプリケーション・ノート

テスト・システム開発ガイド :

- テスト・システム開発ガイド テスト・システム設計入門 (AN 1465-1)、
カタログ番号5988-9747JA
<http://cp.literature.agilent.com/litweb/pdf/5988-9747JA.pdf>
- コンピュータI/Oについて (AN 1465-2)、
カタログ番号5988-9818JA
<http://cp.literature.agilent.com/litweb/pdf/5988-9818JA.pdf>
- ドライバおよびダイレクトI/Oについて (AN 1465-3)、
カタログ番号5989-0110JA
<http://cp.literature.agilent.com/litweb/pdf/5989-0110JA.pdf>
- システムのハードウェア・アーキテクチャと測定器の選択 (AN 1465-5)、
カタログ番号5988-9820JA
<http://cp.literature.agilent.com/litweb/pdf/5988-9820JA.pdf>

- ラックとシステム・インターコネクトの影響について (AN 1465-6)、
カタログ番号5988-9821JA

<http://cp.literature.agilent.com/litweb/pdf/5988-9821JA.pdf>

- システム・スループットの最大化とシステム設置の最適化 (AN 1465-7)、
カタログ番号5988-9822JA

- メンテナンスについて (AN 1465-8)、
カタログ番号5988-9823JA

<http://cp.literature.agilent.com/litweb/pdf/5988-9823JA.pdf>

T&M Programmers Toolkit (AN 1409-2)、
カタログ番号5988-6617JA

その他のリソース

独自のテスト・システムの構築

<http://www.agilent.co.jp/find/buildyourown>

アジレント・デイベロッパ・ネットワーク (ADN)

<http://agilent.co.jp/find/adn>

www.agilent.com



電子計測UPDATE

www.agilent.com/find/emailupdates-Japan

Agilentからの最新情報を記載した電子メールを無料でお送りします。

この文章内の製品仕様と説明は、予告なしに変更される場合があります。

Microsoft、Windows、Windows NTおよびVisual Studioは、Microsoft Corporationの米国登録商標です。

MATLABは、The Math Works, Inc.の米国登録商標です。

アジレント・テクノロジー株式会社
本社 〒192-8510 東京都八王子市高倉町9-1

計測お客様窓口

受付時間 9:00-19:00

(12:00-13:00もお受けしています。土・日・祭日を除く)

FAX、E-mail、Webは24時間受け付けています。

TEL ■■ 0120-421-345
(0426-56-7832)

FAX ■■ 0120-421-678
(0426-56-7840)

Email contact_japan@agilent.com

電子計測ホームページ
www.agilent.co.jp/find/tm

- 記載事項は変更になる場合があります。
ご発注の際はご確認ください。

Copyright 2004
アジレント・テクノロジー株式会社



Agilent Technologies

July 14, 2004
5988-9819JA
0000-00DEP