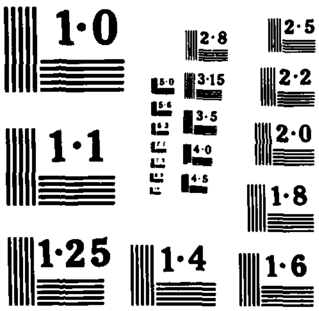
AD-A156 819 LET'S GET SERIOUS - ABOUT SOFTWARE(U) AIR COMMAND AND STAFF COLL MAXWELL AFB AL J R HEGLAND APR 85 ACSC-85-1130 1/1

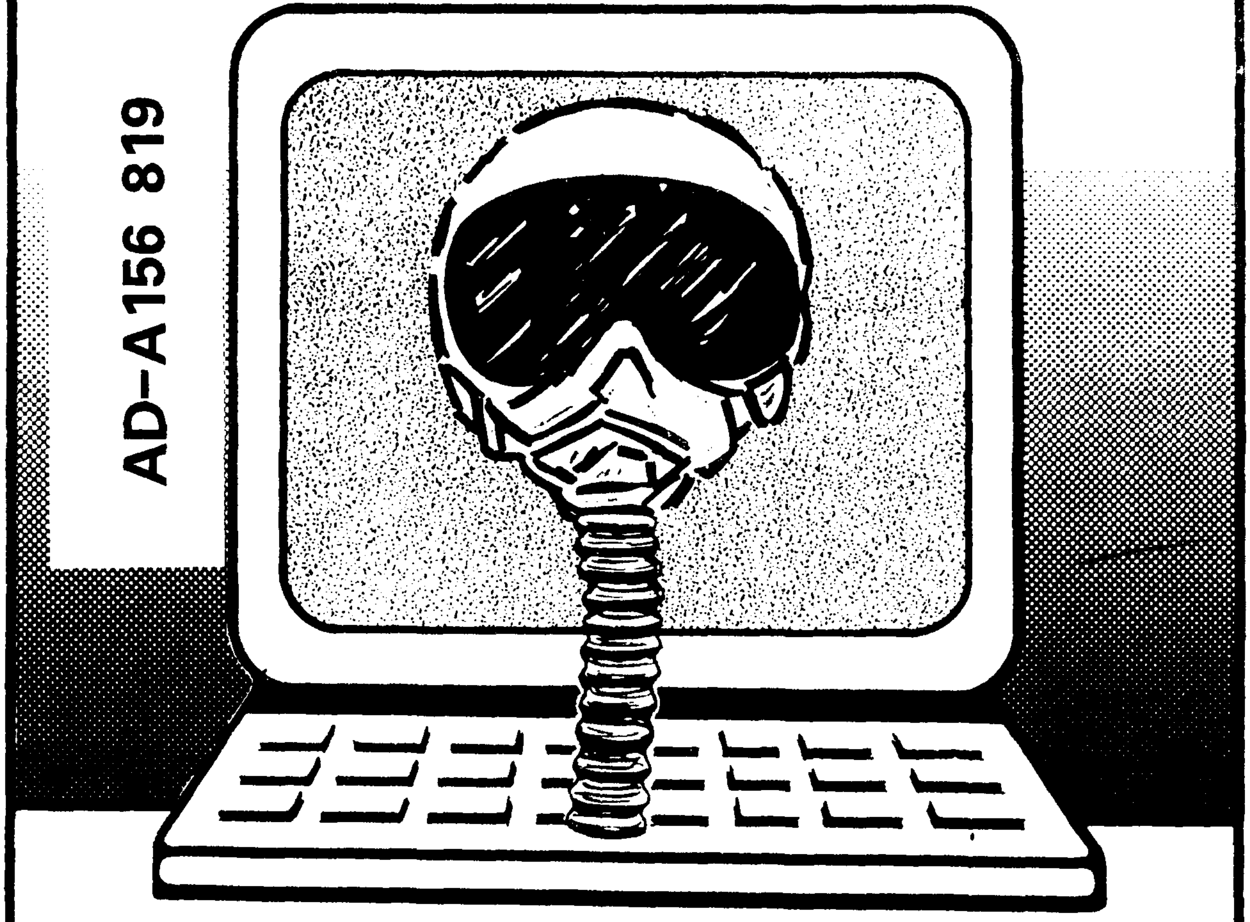
UNCLASSIFIED F/G 9/2 NL

END
DATE
FILMED
9 85


1·0
2·8
2·5
5·0
3·15
2·2
3·5
1·1
4·0
2·0
4·5
1·8
1·25
1·4
1·6

2

Let's get serious . . .



ABOUT SOFTWARE

A Handbook for . . .

TAC FIGHTER SQUADRON SOFTWARE DEVELOPMENT

DISCLAIMER

The views and conclusions expressed in this document are those of the author. They are not intended and should not be thought to represent official ideas, attitudes, or policies of any agency of the United States Government. The author has not had special access to official information or ideas and has employed only open-source material available to any writer on this subject.

This document is the property of the United States Government. It is available for distribution to the general public. A loan copy of the document may be obtained from the Air University Interlibrary Loan Service (AUL/LDEX, Maxwell AFB, Alabama, 36112) or the Defense Technical Information Center. Request must include the author's name and complete title of the study.

This document may be reproduced for use in other research reports or educational pursuits contingent upon the following stipulations:

-- Reproduction rights do not extend to any copyrighted material that may be contained in the research report.

-- All reproduced copies must contain the following credit line: "Reprinted by permission of the Air Command and Staff College."

-- All reproduced copies must contain the name(s) of the report's author(s).

-- If format modification is necessary to better serve the user's needs, adjustments may be made to this report--this authorization does not extend to copyrighted information or material. The following statement must accompany the modified document: "Adapted from Air Command and Staff Research Report ___(number)___ entitled ___(title)___ by ___(author)___."

-- This notice must be included with any reproduced or adapted portions of this document.

REPORT NUMBER 85-1130

TITLE Let's Get Serious . . . About Software

AUTHOR(S) MAJOR JAMES R. HEGLAND, USAF

FACULTY ADVISOR MAJOR ROBERT WEIS, ACSC/EDX

SPONSOR LT COL ROSS L. SMITH, HQ TAC/IGIO

Submitted to the faculty in partial fulfillment of
requirements for graduation.

AIR COMMAND AND STAFF COLLEGE
AIR UNIVERSITY
MAXWELL AFB, AL 36112

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) 85-1130	5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION ASCS/EDCC	6b. OFFICE SYMBOL *(If applicable)*	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS *(City, State and ZIP Code)* MAXWELL AFB AL 36112		7b. ADDRESS *(City, State and ZIP Code)*
8a. NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL *(If applicable)*	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

8c. ADDRESS *(City, State and ZIP Code)*	10. SOURCE OF FUNDING NOS.			
	PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO	WORK UNIT NO
11. TITLE *(Include Security Classification)* Let's Get Serious . . . About Software				

12. PERSONAL AUTHOR(S)
Hegland, James R., Major, USAF

13a. TYPE OF REPORT	13b. TIME COVERED FROM ________ TO ________	14. DATE OF REPORT *(Yr., Mo., Day)* 1985 April	15. PAGE COUNT

16. SUPPLEMENTARY NOTATION

17.	COSATI CODES		18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
FIELD	GROUP	SUB. GR.	

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

→ This guide translates applicable Air Force and TAC data processing requirements into language and examples fighter crews can relate to and understand. It provides information on methods of programming, documenting, and submitting software to TAC. It is designed to help get fighter squadron software into TAC wide distribution and prevent wasted efforts.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. □ DTIC USERS □	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL ACSC/EDCC Maxwell AFB AL 36112	22b. TELEPHONE NUMBER *(Include Area Code)* (205) 293-2483	22c. OFFICE SYMBOL

Once you open a can of worms,
the only way to recan them is
to use a larger can ZYMERGY'S FIRST LAW

PREFACE

STOP – Have you written any computer programs for your unit? Did you know you are required by AFR 300–3 to send **any significant** software program including all programs over **200 lines** to your MAJCOM Small Computer Technical Center within **30 days** after completing it? Did you know that includes programs you may have modified too? Did you know the Tactical Air Command (TAC) Small Computer Technical Center (SCTC) will make your software available TAC–wide? Did you also know that if your program and software don't meet the requirements specified in TACP 300–11, it won't get into the TAC Small Computer Software Catalog? But wait. Don't start your best *guns defense* yet.

This handbook provides a foundation for understanding software development. It can be used in several ways to suit your needs. If you have not developed a program and are interested in doing so, start at the beginning and *press the attack*. You will not become a seasoned programmer, but will understand the methodology of software development. Then, you will be ready to select and learn a computer language, then translate problems into solutions with the aid of the computer. If you have a finished program, refer to Chapter Four and Appendix One to ensure your documentation is up to speed. Chapter Five will help you understand the process involved in submitting your package to TAC. By the way, if you happen to be a fighter squadron operations officer or commander, you can use this guide to understand and manage your unit small computer programmers.

Ok – Now that you have the Rules of Engagement:

You're cleared in hot.

ABOUT THE AUTHOR

Judgement comes from experience,
experience comes from poor judgement
ROBERT E. LEE'S TRUCE

Major James Hegland had his first introduction to computers in November 1966. He attempted to pursuade a Control Data 3600 to convert miles per hour to kilometers per hour. The results were expressed in hours per hour. Later as an engineering student, after many long winter nites at the computer center, he was able to employ the IBM 360, in ways yet to be repeated (and he's not talking either). As a ROTC graduate from North Dakota State, he eventually became a Weapon Systems Officer in the F-4, logging over 1200 hours in *Big-Ugly*. He has been a pogue, instructor, and evaluator.

While in the Republic of Korea, he was the Chief of the 51st Tac Fighter Wing Scheduling Shop. He developed a computer program that presented the wing's flying hours and sortie effectiveness to the House Armed Services Sub-Committee during their fact-finding tour in 1984.

When he PCSed, the flying hour computer program was "trashed" because no one else could get it to run. He vowed not to let that happen again.

He wrote this handbook as a partial atonement for his past programming transgressions. He is a graduate of the Air Command and Staff College, Class of 1985.

When all else fails,
read the instructions
CANN'S AXIOM

TABLE OF CONTENTS

LIST OF FIGURES

Always draw your curves,
then plot your data
FINAGLE'S SECOND RULE

Any sufficiently advanced technology is indistinguishable from magic CLARK'S THIRD LAW

CHAPTER ONE INTRODUCTION

NOTES:

WHAT? ANOTHER BOOK ON SOFTWARE?

TAC was on the leading edge in acquiring small computers during the 1970's. Small computers were to help fighter squadrons automate and streamline flight planning, scheduling, and weapons delivery computations. Unfortunately, the commercial products (software) needed to perform these important tasks were not available. In the typical TAC fighter squadron, "home-grown" programs built by unit small computer programmers were only a partial solution. Poorly organized, or with little or no documentation, many home-grown programs became useless when the originator departed station. Air Force data processing regulations were developed to prevent such "wasted efforts." However, many fighter squadron small computer programmers were unaware of the Air Force data processing regulations and requirements.

In response, TAC produced a pamphlet, TACP 300-11, outlining and standardizing the requirements for writing and documenting squadron home-grown software. The pamphlet, well written by Air Force data processing standards, was not *fighter pilot friendly*. The critical guidance for producing quality software was not fully communicated.

This guide translates applicable Air Force and TAC data processing requirements into language and examples that fighter crews can relate to and understand. It provides information on methods of programming, documenting, and submitting software to TAC. It is designed to help get the software you create into TAC-wide distribution and prevent wasting your efforts.

This guide is exclusively designed for the small computer programmer in TAC fighter squadrons. It is as an ***area munition***, targeted for members of TAC fighter **squadrons** with varied programming experience. It is intended for those with very little experience and those who are experienced programmers.

WHERE'S THIS ALL GOING?

Chapter One provides an overview and defines **software**, **program** and other terms used in this guide. Chapter Two introduces life-phases of programs and software and the process of developing software, using a squadron scheduling scenario as an example. It is important to flowchart and design in maintainability -- you find out why in Chapter Two.

Chapter Three describes the three most popular methods of computer programming and uses a typical flightline example -- calculating takeoff data. Chapter Four introduces documentation techniques and reviews several aspects of technical writing to help you describe your program in clear, concise, meaningful terms.

Chapter Five describes how to get unit developed software into the system for TAC-wide use. The process to request commercially available software is also described in Chapter Five. Chapter Six summarizes the handbook.

You should check your documentation with the requirements outlined in Appendix One before sending it to TAC. Appendix Two is a guide to current Air Force and TAC regulations that applies to the fighter squadron computer programmer. Commanders, operations officers, and squadron small computer programmers should review them before beginning any software development efforts.

A reader's response form, the last page of this handbook, is your opportunity to comment on the helpfulness of this handbook. When you're through -- rip it out and send it to TAC/IGIO with your comments.

Now, a few definitions to standardize terminology. The first two are: **program** and **software**. The next segment should provide a basic understanding of the terms and their use.

WHAT ARE PROGRAMS?

A **program** is a list of instructions that control the computer. Programmers write these instructions in **source code**, a specific type of computer "language". The languages include BASIC, COBOL, FORTRAN, Ada, and others. The terms source code and program are often used interchangeably.

A program is like a flight scheduler. Just as the scheduler says when and where you'll fly, a program tells the computer when and where to process information. The scheduler takes **input** in the form of crew names, aircraft, and airspace, and **processes** it to produce an **output**: sorties, hours, and training. A program is a set of written instructions designed to take **input**, **process** it, and create **output** in the same manner.

THEN, WHAT IS SOFTWARE?

The term **software** refers to computer programs. A more correct definition includes the **program**, the **data**, and the associated **documentation** used in the operation of a computer. **Data** is the information put in the machine. It may be a list of aircrew names and qualifications, birth dates, or checkride eligibility dates. **Documentation** is the complete set of associated guides, user manuals, checklists, and diagrams needed to use or maintain the program. It is like the Dash One and associated technical orders needed to fly and maintain your aircraft.

TYPES OF SOFTWARE

There are three types of software: **system**, **support**, and **applications**. **System software** comes with the computer when you buy it. Called the Operating System, it has only one function -- make the computer work. **Support software** aids in developing new programs and includes flowcharting and documentation routines to make life easier for programmers.

This handbook concentrates on **applications software**. Typical applications may include flight planning, scheduling or weapons computation. TAC further divides applications software into **safety-of-flight**, **non-safety-of-flight**, and **unit-unique** software (these categories are further described in Chapter Five). Applications software gets the job done, and is usually written by people like you (squadron navigation officer, scheduler, weapons officer, etc.), who are familiar with the tasks the computer will assist once programming is complete.

SUMMARY

Programs were developed in the TAC fighter squadron in response to a need, but many programs went unused when the originator ***departed the fix***. If you write a program, you are obligated (by Air Force regulation) to send it to your MAJCOM. This handbook concentrates on software development, and outlines how you can get your program and documentation (software) **approved** and **distributed** by TAC. The next chapter starts you on the way to **organized** software development.

If you choose Modular Programming, you'll find the logic is easy to develop and understand. Complex problems are divided into simple, manageable elements.

STEPS FOR MODULAR PROGRAMMING

Divide the program into segments.

Ensure modules perform only one task.

Check for cohesion and independence.

Use "calls" from the mainline routine.

Ensure each module has only one entry.

Ensure each module has only one exit.

You can reuse satements by "recalling" the routine when needed. You can also create a **library** of modules and use in other programs. Modular programming allows other users to easily unravel and understand your program. It is easy to maintain and errors are easily traced and fixed.

SUMMARY

When you use an organized method of programming you should pick one that fits your *style*. The three most popular methods are Structured, Top-Down, and Modular. They produce software that is understandable, reliable, and maintainable.

STRUCTURED PROGRAMMING: Approaches programming as a sequence of segments made up of instructions. It uses three logic elements: the sequence, choice and loop. The steps are:

Divide the problem into segments

Sequence the instructions so each:

Perform only one specific task.

Complete each task, then continue.

Have only one entry/one exit.

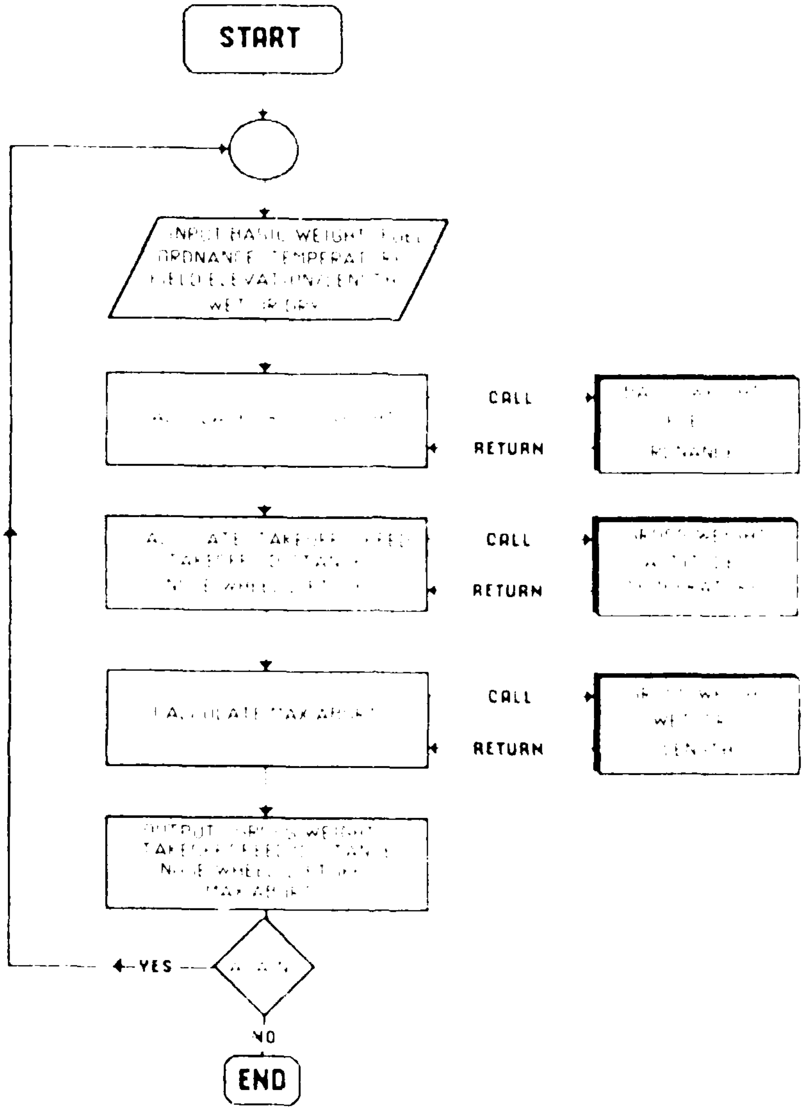


Figure 7. Modular Programming of Takeoff Data

MODULAR PROGRAMMING

If you prepare a mission briefing and assign all navigation tasks to one individual, all target area tactics to another, and the briefing slide preparation to a third, you establish modular activities. **Modular** programming is a technique used frequently for designing and writing long, complex computer programs.

Like Structured and Top-Down, Modular programming evaluates the problem and divides it into small segments. The instructions in each segment perform only one task. For example, the **takeoff** section of the Dash One has charts to perform only one task and forms a **module**. Modules are often refered to as **subroutines**. Subroutines are small programs within a larger program.

All statements in a module should be closely related to each other, or **cohesive**. But, each module should be **independent** of other modules so when modified or changed, one module won't impact the statements or logic of other modules.

When you delegate briefing preparation tasks to members of your flight, you remain the flight lead. You are responsible for directing and tying the briefing together. The main program directs the flow of logic through the modules. It uses **call** statements to activate the other modules (or subroutines). When the operation in that module is completed, the "**return**" instruction transfers action back to the main program. Just as you are the point of contact for your flight, each module has only one entry and one exit.

Now, use Modular programming techniques and determine the segments (input, process, output) for the takeoff data example. Divide the segments until only a single task is accomplished within a module. You can identify one subroutine as Gross Weight and "call" it from the main program.

The Takeoff Speed, Distance and Nose Wheel Liftoff are subroutines and can be "**called**" from the main program. The Max Abort calculations are "called" likewise. The Dash One tables are represented by the information used in calculating the answers. For the exercise in programming, draw the flowchart and check your answer with Figure 7.

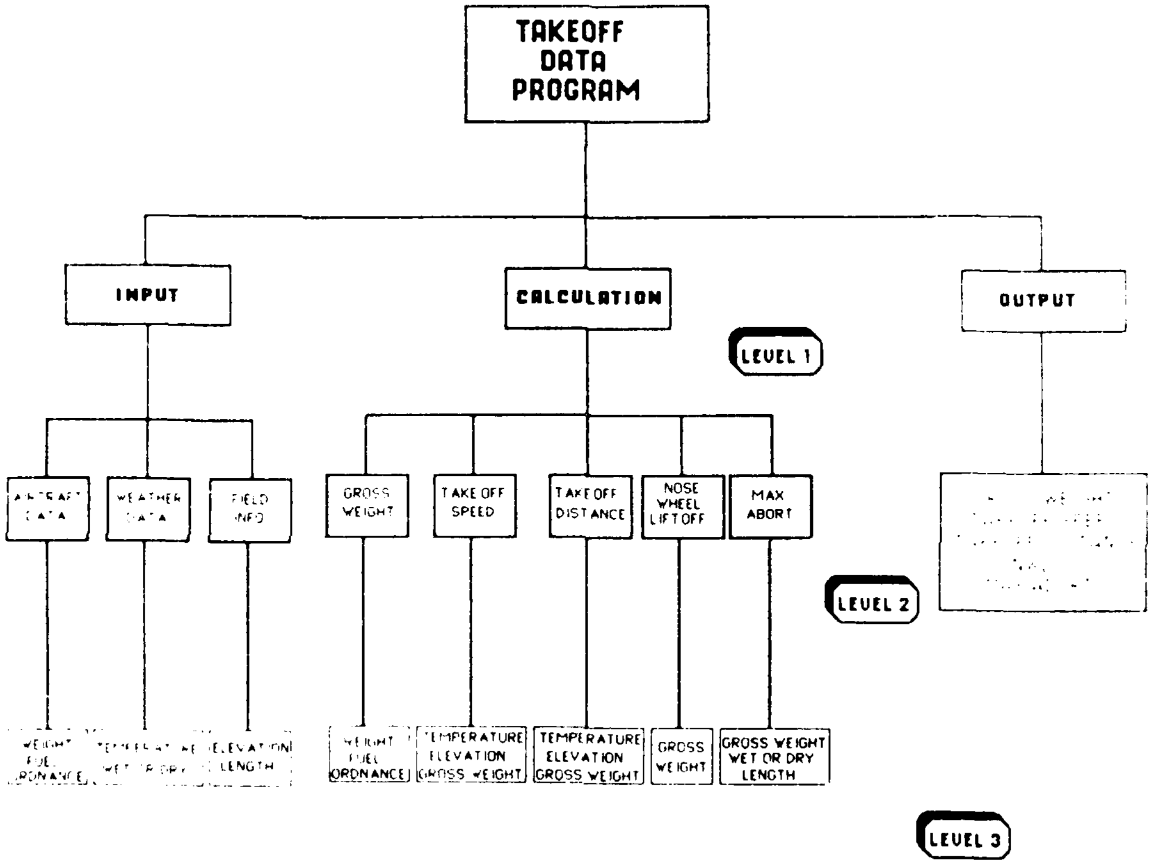


Figure 6. Top-Down Programming of Takeoff Data

TOP-DOWN PROGRAMMING

When planning a complex mission, you might first outline the overall events and then begin the detailed work. With several crew members assigned to a mission, you can split up the tasks. The **Top-Down** method also divides and conquers a problem. A difficult problem is reduced to smaller components.

Top-Down programming designs the program in stages. The flowchart looks like an organizational chart. The top levels oversee the lower levels. When a Top-Down program is run, all the details at the lowest level of the first branch are completed, then the next higher level of the same branch is done. All tasks in the first branch are accomplished, then the next branch starts at the lowest level. When running, the flow of a Top-Down program is left-to-right, bottom-to-top. Results are "fed uphill" as in the wing organization. **Note**: When using Top-Down, you don't have to use the ANSI standard flowchart symbols.

STEPS OF TOP-DOWN PROGRAMMING

Design in levels: overview first - then details.

Design first, then select a programming language.

Save all the detail work for the lower levels.

Check each level as its written.

Identify the broad functions or tasks of the program. For example, basic program functions used in calculating takeoff data are: (1) **Input**, (2) **Calculation**, (3) **Output**. These branches form Level 1 of your flowchart. Divide these functions into smaller, more detailed subfunctions. Level 2 elements, for the **Input** branch, include the gross weight, temperature, and field information. Calculating the Takeoff Speed and Distance is also on Level 2 - the **Calculation** branch.

Level 3 supports Level 2. For example, USAF Form 365F, fuel, and mission ordnance inputs are subordinate to Level 2 gross weight input information. The weather briefing, and the field information also support Level 2 inputs. For an exercise in Top-Down, try and draw the flowchart. If your answer is like Figure 6, you've got Top-Down in your sights.

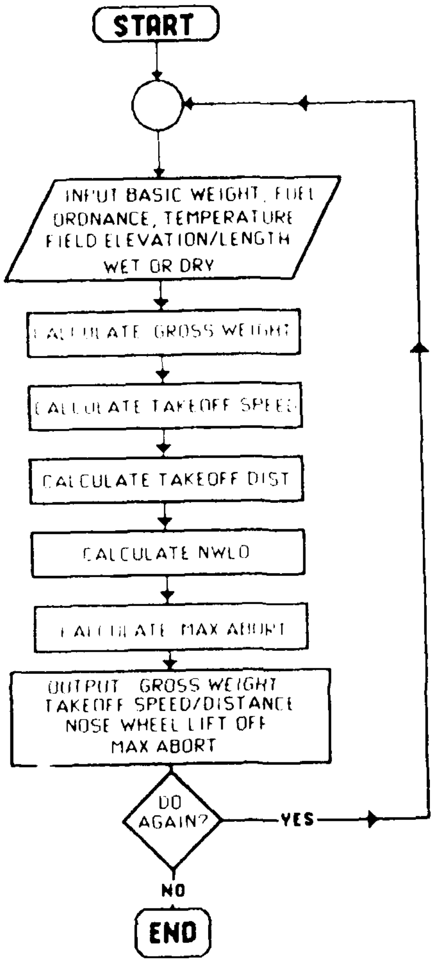


Figure 5. Structured Programming of Takeoff Data

When you write a program for the line-up card, assess the input and output requirements, then develop the process.

INPUTS

Aircraft	**basic weight, fuel, ordnance**
Weather	**pressure and temperature**
Environment	**runway length, elevation, wet or dry**

OUTPUTS

Gross Weight
Takeoff Speed
Takeoff Distance
Nose Wheel Liftoff Speed (NWLO)
Max Abort Speed (wet or dry)

Aircraft basic weight (from the USAF Form 365F Aircraft Weight and Balance) with fuel and mission ordnance added gives Gross Weight. The weather information (from the DoD Form 178-1) provides the temperature, density, and pressure information. Field information is in the Flight Information Publications (FLIP). Normally you use the Dash One to calculate takeoff data, but for this exercise in programming, write out the sequence of logic and draw a flowchart. Don't worry about the details of Dash One formulas. Keep the flow from start to finish, one task at a time.

When using the structured programming method, determine the steps the segments will depict, then use the three basic elements: sequence, choice, and loop, to organize the solution. An example of a Structured Programming solution of the takeoff data problem is in Figure 5.

Takeoff data calculation is easily programmed into a structured format. This method breaks a problem into logical sections and simplifies programming by using three basic elements: sequence, choice, and loop. The benefits associated with the Structured Programming method are it:

Encourages programming discipline.

Has fewer logic errors.

Is easily modified and maintained.

STRUCTURED PROGRAMMING

Many of the first fighter squadron "home-grown" small computer programs were "pasted" together using ANSI flowchart symbols with little regard to logical organization. The result -- many programs were difficult to read, understand, or maintain for anyone other than the creator. In the late 1960's a standardized method was developed -- **Structured Programming**.

Structured Programming concentrates on one of the most error-prone factors in programming -- logic. A structured program is made up of segments (sets of instructions written in source code) executed from start to finish. This method simplifies most complex programs and makes them more readable and understandable. They are simple to develop, maintain, and modify. The logic of Structured Programming uses three basic elements: the **sequence**, **choice**, and **loop** (Figure 4.).

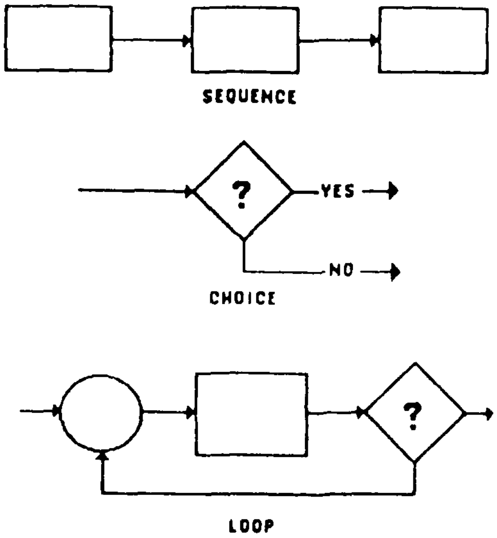


Figure 4. Structured Programming Elements

In a structured program, each segment is a set of instructions that:

Performs only one specific task.

Completes each task then continues.

Has only one entry/exit.

CHAPTER THREE SOFTWARE DEVELOPMENT

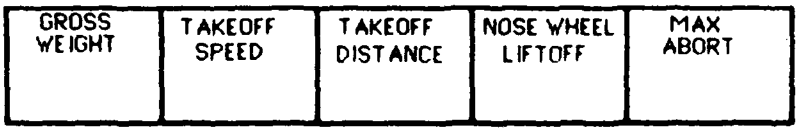
The amount of expertise varies in inverse proportion to the number of statements understood by the general public GRUMMIDGE'S LAW

NOTES:

This chapter introduces the three most popular methods of computer programming: **Structured**, **Top-Down**, and **Modular**. The programming method you choose influences the structure of the proposed solution. All three methods are efficient and logical. They produce software designed for easy understanding, maintenance and reliability. Organized methods help prevent errors, and if errors occur, they will be easier to find and fix. Review all three, and pick one that suits your *style*.

TAKEOFF DATA EXAMPLE:

All aircrew members fill out a line-up card with mission takeoff data before each flight (Figure 3).



GROSS WEIGHT	TAKEOFF SPEED	TAKEOFF DISTANCE	NOSE WHEEL LIFTOFF	MAX ABORT

Figure 3. Line-Up Card, Takeoff Data

Calculating takeoff data is an example familiar to all aircrew members and illustrates the process of developing a computer program. You must **define** the problem, **organize** the solution, **write** the code, **test**, and **document**.

The problem is easily defined -- calculate the takeoff data using a computer. You can organize the solution with one of many programming methods and outline it with a flowchart. In the next few pages, three "tried and true" methods illustrate the process.

The circle is a connector. It takes several paths, represented by the arrows, and ties them together like a common turn point for two separate low level routes. The box represents a process. The oval is used at the beginning or end of a flowchart. The slanted box represents an input. The diamond shape is a decision, with a yes or no path leaving the shape

These symbols can be used as you identify and structure the proposed solution to a problem. When you use these standard symbols and flowchart, you have taken an important step toward an organized and documented software.

SUMMARY

Software has three life phases. The most obvious are **development** and **use** phases. **Maintenance** is the most neglected aspect. Anytime new features are added or modifications are made to a functioning program you are maintaining it. A program that can be maintained, survives the inevitable changes and future adjustments. How well you develop your software determines how well others will use and maintain it. The key is using an organized process.

An organized process helps develop useable, maintainable (quality) software. First **define the problem**, then **organize the solution**. Once organized, **write out the solution** in source code. Next, **test the program** by desk checking the logic and running it on the computer. Include realistic data in your testing, and ensure the results are accurate. Finally, **complete the documentation**.

A flowchart helps develop the solution to a problem. You should develop your program with an organized method. The simplest methods are: Structured, Top-Down, and Modular. These methods are outlined in the next chapter.

FLOWCHARTING

An organized approach will help you develop quality software. A flowchart helps you get organized. A flowchart is a map of what the program is going to do and shows how it is going to do it. Think of it as the grease board in the scheduling shop. Both are used to get thoughts organized, and once organized, written down in final form. The schedule gets printed -- the program entered into the computer.

Just as the scheduling grease board in one squadron is different from another squadron, no two programmers use the same technique. There are several recognized flowchart symbols. The American National Standards Institute (ANSI) identifies the symbols in Figure 2 as standards to use when flowcharting. These symbols are available on templates to make drawing flowcharts easier than a "cut and paste" low level mission.

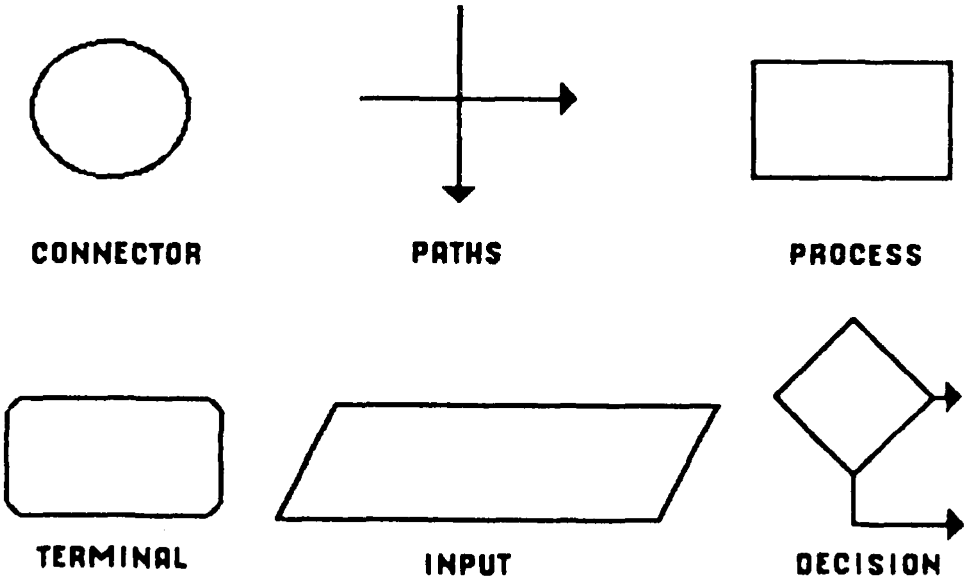


Figure 2. ANSI Flowchart Symbols

When organizing software or a program, you must resolve the same questions. You sequence the events and determine step-by-step actions to solve the problem.

Write it out.

After the schedule is outlined and the sequence of events determined (airlift, maintenance, sortie and crew requirements), you print it on a scheduling form. The draft schedule gives you a starting point.

As a programmer, you write the program instructions in the computer language (code) best suited to the problem. Like the draft schedule, you test it before you start using it.

Test (Check it out).

A successful scheduler checks and rechecks the schedule before "going to print". You might consult with another scheduler, a flight commander, or assistant operations officer. Then, having incorporated the appropriate changes, take it to the commander.

Software requires the same degree of checking as the draft schedule. First "desk check" it. Desk checking is reading the code line-by-line to see if it follows the logic of the **organized** solution. Once satisfied with the desk checking, run it on the computer and have other operators run it. Test the data and be sure the results are accurate. When the program is accurate -- finish the documentation.

Document your results.

A smart scheduler keeps a notebook as the deployment schedule develops to record all lessons learned along the way. You can use the notes when writing the "after action" report and continuity folder.

When you begin developing software, start taking notes. Documentation is an after action report and a continuity folder. If you want to get your program into the TAC system -- it must be documented.

A valuable tool for organization is the flowchart. It helps get your thoughts in sequence and is often helpful when describing and defining the problem.

FIVE STEPS TO SUCCESS

Developing software is like building a flying scheduling program. Both are organized processes and use the following steps:

Define the problem
Organize the solution
Write it out
Test (Check it out)
Document your results

The five step process of developing software may be represented by a sequence (Figure 1).

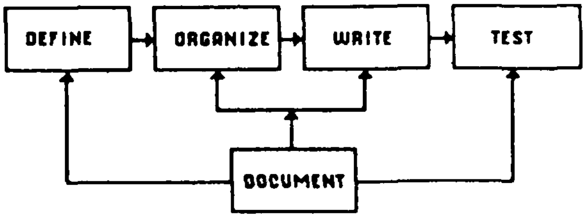


Figure 1. Software Development Process

Define the problem.

If your commander tasked you to plan (develop) a deployment to Base X, then the flying schedule and the redeployment, you would have a few questions to answer. For example: How many sorties will be flown? How many hours available? How many aircraft available? Has it been done before? Can a previous approach be applied?

As a programmer and software developer, you answer similar questions at the outset of the project. Analyze the problem, determine the specific input, the processing, and output needed for the application. You should check with the wing Small Computer Manager (SCM) to see if software exists which may be used as is or modified to solve the problem.

Organize the solution.

While working the deployment, you determine types of sorties and the sequence of mission events, considering crew rotation and support requirements, airlift, maintenance, etc.

CHAPTER TWO
SOFTWARE DEVELOPMENT

If builders built buildings the way
some programmers write programs,
the first woodpecker that came along
would destroy civilization WEINBERG'S LAW

NOTES:

This chapter introduces three life phases associated with programs and software. The phases are: **development**, **use**, and **maintenance**. You must consider these phases when creating software. This chapter also introduces an organized method of developing software and emphasizes the importance of using flowcharts

SOFTWARE LIFE PHASES

Imagine yourself as the squadron scheduler. In the **development** phase of the schedule you structure flying activities on the scheduling grease board and eventually on a printed form. You do the same thing when developing software. First the flow and structure of the program is laid out, then the code is written and entered into the computer.

With the schedule complete: crews, aircraft and missions aligned, it is used to produce sorties, hours, and training. When you run your program, manipulate data, or make a printout of the output, you are in the **use** phase.

Most often forgotten is the **maintenance** phase. A scheduler holds options and anticipates possible changes. A good programmer anticipates reconstruction and refinement of software. When you add new code or routines to a program, improve it's features, or make it run smoother, you are maintaining it.

How do you design in maintainability? How do you ensure the final product is easy to use? The answer to both of those questions is in the **development** phase -- how you develop your software. The secret is -- use an organized process. The next section presents five steps to help you successfully **develop useful** and **maintainable** software.

TOP-DOWN PROGRAMMING: Top-Down also looks at a program as a series of segments in a tree-like fashion, like a wing organizational chart. The steps for Top-Down:

Design in levels: overview first - then details.

Design first, then select a programming language.

Save all the detail work for the lower levels.

Check each level as its written.

MODULAR PROGRAMMING: Modular uses subroutines to ease the task of the programmer. Instructions are written in segments that perform only one task. You may include pretested routines from a library. The modular programming steps:

Divide the program into segments.

Ensure modules perform only one task.

Check for cohesion and independence.

Use "calls" from the mainline routine.

Ensure each module has only one entry/ one exit.

Pick one of the three methods and, like the scheduler in the deployment example, keep notes as you develop your program. With the notes -- begin documentating your program. This chapter organized your programming. The next chapter provides clues you can use to create organized documentation.

CHAPTER FOUR
DOCUMENTATION

An ounce of image is worth
a pound of performance
NOLAN'S PLACEBO

NOTES:

This chapter describes techniques for creating documentation. Software documentation communicates factual information to the program user. Your documentation may be TAC's introduction to you as a programmer. To sell your solution and get TAC-wide distribution, you must document well. You produce good documentation by using two major rules:

Plan from the beginning.

Anytime you sit down to program, organize and outline your thoughts, intentions and direction. Write them down -- keep notes.

Make it readable.

Your documentation will be better understood if the language and writing style are based on good technical writing standards.

Documentation comes in many forms, but may be broadly categorized as **external** or **internal**. External documentation includes **user's manuals**, **reference cards**, and **technical manuals**. External documentation helps during use and maintenance of your software. Internal documentation includes **screen menus** or internal **comments** and **remark** statements. **Screen menus** help the user through the steps of a program. **Comments** or **remarks** are statements built into the program. These statements communicate with the maintainers and modifiers of your program.

MANUAL ORGANIZATION

The biggest mistake you can make with a user's guide or technical manual is to "not plan the project." The organization must make sense to the user. Each organizational element of a manual has a specific purpose. The main purpose is giving the user instructions. Giving instructions rests on three key points:

You must understand the process you are instructing.
You must communicate the sequence of instructions.
You must pick words your audience can understand.

When you organize you should consider: **relevance**, **sequence**, **balance**, and the **format** of the documentation.

RELEVANCE

What role does it play? Is your target audience made of experienced or novice users? The answers influence the type of writing, organization, and level of detail in the manual. Include only related or pertinent information in the manual.

SEQUENCE

Arrange the material to meet the needs of the users. You may need to sequence information based on input or processing requirements of the program. Sequence appropriately.

BALANCE

The amount of material should not vary excessively from one operating feature of the software to another. Don't shortchange important subjects because you think most users will not use a feature or an operation *seems* obvious to you.

FORMAT

Use different type faces or fonts (bold, italic, underlined), to give the reader variety, but be consistent. Make all chapter headings, subsections, and organizational elements consistent with each other. Arrange text on the page to highlight important points. Leave plenty of room for users' notes. Make your solution accurate *and* attractive.

WRITING STYLE AND READABILITY

Your style and skill impact the readability of your documentation. If a user must read the manual to make the program work, then the harder it is to read,the harder the program will be to use. Skillful writing is clear, coherent, and concise. Your manual must reflect these qualities, or you fail your readers. Make the instructions interesting and relevant to the user.

Here are a few tips for writing documentation. Air Force Pamphlet AFP 13-2, *Communicating To Manage in Tomorrow's Air Force (The Tongue and Quill)*, is a great reference for "polishing up" your writing skills.

Get Organized. Before briefing a mission, you get organized. You must first start with a plan – then communicate that plan. Software documentation follows the same process as a mission briefing. You use topic sentences and overviews to communicate the plan to your reader.

Choose the right wording. Don't try to dazzle the reader. You are writing for fighter crews, so keep the words short, familiar, and to the point. Use common words instead of computer jargon, acronyms, or symbols.

DON'T USE:	USE:
component	part
facilitate	help
terminate	exit
initialize	start
purge	erase
minimize	reduce
bug	error
power down	turn off
%	percent
hex	hexidecimal
lpm	lines per minute
RAM	random access memory
boot	start up
interface	connection
utilize	use

Pacing. When giving a mission briefing, you don't go into detail on the formation landing and skimp on the area tactics. When you get to the really tough stuff in your manual -- take your time. Don't spend four pages on how to insert a disk into the disk drive, then gloss over the real "meat" of the program.

Comparisons and Examples. Relate to what your readers know. Use **similes**, explicit comparisons using the words **as** or **like**. For example, when you put a disk into the disk drive, it's like putting bread into a toaster. You can use **analogies**, describing one thing by drawing comparisons to another. Describing the pacing of a mission briefing as similar to pacing in a manual is an analogy. A third method of making a comparison is the **metaphor**. The metaphor is a comparison you imply rather than state outright. To say your program is, *'smarter than the average wingman'*, is a metaphor.

Tone How you to express yourself in the manual is **tone**. A "friendly tone" helps communicate the instructions and is less formal For a less formal tone:

> **Give instructions as if you were standing there - talk to your user.**

> **Be sympathetic and helpful. Accept responsibility for how well they use the manual.**

> **Write in the second person. Use "you" and "your" throughout the manual for friendlier tone.**

Voice. Keep your writing clear and direct. If you use "passive voice" it creates lifeless, hard to read instructions. Write in active voice. Put the subject before the verb and the object after. Compare:

> **"The program may begin after inserting the disk into the primary drive unit."**

with

> **"Put the disk in drive A."**

Both give the same instruction, but which is more direct, vital, and **alive**?

Tense. Keep your text simple. Use consistent verb tense. If you continually shift from present to past tense, the readers will lose their train of thought.

Attention to Detail. Be consistent. Don't use **monitor** in the first half and then change to **video display**. Don't spell **disk** one way then try **disc** later on. Don't use abbreviations until after you've defined and identified them. Once identified use only the abbreviation. Proofread, proofread, proof~read your draft.

Grammar and punctuation. Make it clear. Pass your draft copy to the best reader in the squadron or wing. Even if it costs you a beer, the squadron adjutant may catch some of the errors you and other proofreaders missed.

Accept feedback. Listen to your "proofers" and be honest with yourself. Does the text make sense? Then go for it. If not, grit your teeth and *soak up the shot*.

INTERNAL DOCUMENTATION

This section provides a set of rules for creating internal documentation, **screen menus** and **comments** or **remarks** .

SCREEN MENUS

Tie it in with the manual. Any message on the screen should reflect the same information in the manual. Users unfamiliar with the program need to feel that *someone* knows what is going on, since they may not. If you send message s different from the manual you add confusion.

Keep the user oriented. Screens and menus should let your users know where they are in the program. Menus should show how to get from where they are to where they want to go.

Be Reasonable. Don't fill the screen from top to bottom. Confusing menus are threatening. Menus should offer only options actually needed at any one moment. Anticipate your users' requirements.

Avoid violent language. Avoid violent computer jargon. Words like **fatal**, **aborted** and **crashed**, have unpleasant connotations for computers and aircrews.

Be consistent. Keep the terminology standard in the screens and manuals. If you use **Stop** to leave the program, don't change to **Exit** or **Quit**.

Provide feedback. If the program takes time to process data or information, say so on the menu. Let the user know the system is still working and hasn't "frozen up."

INTERNAL COMMENTS AND REMARKS

Comment statements or remarks are internal documentation describing the inner workings of a program. Be careful when you use comments or remarks in your program. **Placement** is *very* important. Put comments near the operation you are explaining. Comments should highlight the logic structure. You can use blank and other characters as **reference** points. This reference helps when you write the external documentation. Comments should contain **useful information**, and promote the design of **maintainable** programs.

SUMMARY

You must make sure the documentation is accurate and makes sense. you are responsible for how well the user understands and operates your program. Well organized and structured documentation really helps.

Technical writing has many options for the choice and sequence of words. You must check all aspects carefully. Proofread the entire package for spelling, grammar, and overall readability. Compare each screen and menu with the manual. If the words don't match exactly, you risk losing the user and your credibility.

Ferform an FCF on the package. Get a practice user to *ops check* the complete software package. What makes perfect sense to you may not be so clear to someone new to the program.

After you are satisfied that it is technically accurate, use the checklist in Appendix One. Be sure your documentation meets TAC's minimum requirements.

The next chapter explains the process you use to get your software package into TAC-wide distribution. Good documentation is an important step.

CHAPTER FIVE
THE TRANSACTION

*Every task takes twice as long
as you think it will take If you
double the time you think it will take
it will actually take four times as long*
DEADLINE-DAN'S DEMON

NOTES:

SOFTWARE EXCHANGE

TAC has a software exchange program that makes unit-developed software available to all TAC units. The program is designed to prevent duplication of effort and *reinventing the wheel* As a software developer you have a piece of the action as well. Once your software package is completed, you are required to submit it to TAC for eventual distribution. The local Small Computer Manager (SCM) will help you get through the process. The name and telephone number of your wing's SCM should be listed in the unit Small Computer Custodian's Continuity Folder. You should check with the SCM when you start the software development process because they are the **first** reviewer of your documentation and program. Work with them as you develop your software.

When your'e done, get all information required to support your program and visit the SCM. You will need a copy of your program in machine-ready format, on 5 1/4 inch diskette. First, the SCM checks the program and documentation for compliance with current TAC directives and regulations. The checklist in Appendix One, will help ensure the package meets requirements.

Once the package is reviewed by the SCM, they forward it to the TAC fighter squadron functional manager, TAC/DOZ. At this point it's all out of your hands. TAC returns a similar quality replacement diskette in exchange for yours, but be sure and keep a file copy as a backup.

The software package is evaluated by the TAC Small Computer Technical Center (SCTC). Documentation is *VERY* important. It is the first thing they check at TAC. If you did a good job you're in there. After the documentation passes the test, the SCTC runs the program. They check the results and read the computer code, line-by-line (make sure your comments are appropriate).

With adequate documentation and results as advertised, your program will be **certified**. The next quarterly issue of the TAC Software Catalog should list your program. As the developer, *you* are responsible for keeping the program current. Send all updates or modifications you make to TAC via the wing SCM.

If your software impacts "safety-of-flight", several extra steps are required before TAC will distribute it. By regulation, even you **cannot** use your program until it is **validated**. The example in Chapter Three, calculating takeoff data, requires the extra steps, since it involves safety-of-flight.

The review process is the same at the wing and MAJCOM level, for both the **safety-of-flight** and **non-safety-of-flight** software. Once the TAC SCTC has reviewed it, it is sent to the USAF Tactical Air Warfare Center (USAFTAWC/DOY), for **validation**. Next, TAC Standardization and Evaluation (TAC/DOV) **certifies** the software package. And finaly, TAC Flying Safety makes a final review of the documentation. If at any point your software package fails, it will be returned so you can correct it. During the process the experts make recomendations to the identified deficiencies. The review process ensures safety is not compromised and the software is maintainable.

HOW TO ORDER SOFTWARE

There are two types of software you can order. The easiest to acquire is software developed by unit user/programmers like yourself. Commercial software may be ordered but, is slightly more involved.

TAC-DEVELOPED SOFTWARE

TAC and other MAJCOM unit-developed software is free. Guidelines for ordering are in the latest edition of the TAC Small Computer Software Catalog and TACP 300-11. Find the **Functional Identification Number** in the catalog and use it in your order. Send the order to TAC SCTC through the wing SCM.

TAC will send a copy of the software on a 5 1/4 inch diskette with all associated documentation. You have 10 working days to copy the materials and return them to TAC.

COMMERCIAL SOFTWARE

Commercial software is ordered using the Mini-Data Automation Request (Mini-DAR). The format is in Appendix Three. Before ordering commercial software, you must justify your requirement. The wing SCM can assist you in preparing the Mini-DAR. You must get the Deputy Commander for Operations to sign it, then forward it to your Numbered Air Force and TAC.

Numbered Air Force reviews your request and makes a recommendation then forwards the Mini-DAR to TAC. There, the request is evaluated by the functional area manager (TAC/DOZ). If it is valid, TAC Data Automation approves it. Once approved, a Data Project Directive is generated, and your order will be processed. If you demonstrated how the software will be cost effective in your operations, you greatly improve your chances of approval.

SUMMARY

The process you use to get your software into the TAC Software Catalog starts with your local SCM. If you need to purchase commercial software, you must convince not only your commander, but the Deputy Commander for Operations, Numbered Air Force, and Headquarters TAC. You must have a convincing Mini-DAR and follow the format. The SCM is the person you start with.

THE BOTTOM LINE:

Always check with your local SCM, before you start.

The solution to a problem
changes the problem PEER'S LAW

CHAPTER SIX SUMMARY

NOTES:

This handbook is designed for the TAC fighter squadron small computer programmer. Its objectives: **improve the quality and reliability of unit-developed software and help get it into TAC-wide distribution**. Remember that a program is a set of instructions that control the computer. Those instructions are designed to take **input**, **process** it, and create useful **output**. Software consists of the **program**, the associated **data**, and **documentation**. To successfully develop software you need to use an organized process, much like building a flying training or scheduling program. The five steps are:

Define the problem.

Plan the solution.

Code the program.

Test the program.

Document the program.

Unfortunately, too many programmers don't organize, flowchart, or document enough. The result -- their good intentions become wasted efforts when they PCS.

Another failure is to not consider that someday, some other eager programmer may see an added application for your program If you correctly designed in maintainability, your efforts will be appreciated by that programmer.

How you elect to describe and organize the solution is based on your "style." But, you need an organized method. Three alternatives are: the **Structured**, **Top-Down**, and **Modular** methods. Each method has its advantages. The key is pick one -- and get organized. The Structured method uses three basic elements: the **sequence**, the **loop**, and the **choice**. Top-Down reduces complex problems into smaller ones and links them in an organizational chart fashion. Modular builds smaller, almost independent, programs that can be called from the main program.

The documentation you develop stays after you PCS. It tells others how to make the program run, what it is supposed to do, and how to fix it. A common problem with too many programs has been poor or nonexistent documentation. You can write **very good** documentation. A few simple rules to get you through the challenge:

1. **Start at the beginning. When you flowchart and organise – start the documentation.**

2. **Write as if you were standing beside your user talking to them. Use the second person, make them feel comfortable.**

3. **Make it readable. Use comparisons your readers can relate to.**

4. **You are responsible for how well your readers can use and understand your product. You must help them.**

5. **Use comments or remarks to aid maintainers.**

6. **Use the checklist in Appendix One.**

Double check the regulations. The summaries in Appendix Two are only for guidance. You need to communicate with the **data processing experts**. Your wing has a Small Computer Manager. TAC has devoted an entire office to assist you. If you have questions start at your wing and work your way to the Small Computer Technical Center. The same goes for submitting your software package or ordering software. Start at your base and "work the system."

Just remember two things when you get tied up in a furball of "datamation" experts:

ONE: The mission is to *FLY* and *FIGHT*.

TWO: Software is a *MEANS* not an *END*.

APPENDICES

APPENDIX ONE

This appendix provides a checklist to ensure your documentation meets the "TAC Standard." This checklist was developed from current TAC requirements. A quick call to the good people in TAC/ADUBS will ensure the information is still valid.

TAC requires the documentation as a minimum address:

GENERAL DESCRIPTION
SYSTEM DESCRIPTION
ENVIRONMENT
PROGRAM MAINTENANCE

The format is not specified. You should use this outline to check your documentation, before you visit the local SCM.

GENERAL DESCRIPTION

PURPOSE OF THE PROGRAM. Give a history of the project and state the requirement the program supports. Don't just give the title of the program, but be brief.

TERMS AND ABBREVIATIONS. List terms, definitions, or acronyms that are unique to the program. Don't include items like names of variables or data code names. Explain those in the body of the documentation. If you have a long list of terms, make an attachment.

PROGRAM USE. Tell how it is to be used. What results should a user expect? Are there options available for output, are there specific requirements to the format of inputs?

PROGRAM OPERATION. Give a step-by-step procedures checklist. How to load, setup, and successfully run your program.

PROGRAM MAINTENANCE/MODIFICATION. Include flowcharts and other logic diagrams as well as a listing of your source code. A good reason to build in **Comments and Remarks**.

SYSTEM DESCRIPTION

SECURITY AND PRIVACY. If your program accesses, generates, or uses classified information, describe the items in this section. If it uses information covered by the Privacy Act, be sure and mention that as well.

GENERAL DESCRIPTION. If your program is part of a larger system or data base, explain how it interacts with the other systems, and limitations, if any.

PROGRAM/SYSTEM DESCRIPTION. Give the details and characteristics of your program. Include all routines and subroutines in the description. Make it easy for another operator to maintain or modify by including information you would need too:

IDENTIFICATION. Give the program title and version number.

FUNCTIONS. Describe the program functions and the methods the program uses to accomplish them.

INPUT. describe the input, including again enough information for maintenance.

Explain the types of data used in operation, and the types of records used.

PROCESSING. Describe the processes performed in your program.

MAJOR OPERATIONS. Include quick reference charts if needed to explain what happens.

RESTRICTIONS. Did you build in any restrictions? If you did, explain them.

EXIT REQUIREMENTS. Do you need to close files before leaving? How do you exit? Let your users know how to quit the program and not lose any of their hard work.

STORAGE. How much and what type is needed.

OUTPUT. Describe the types of output your product generates. **Screen Dumps** are *VERY* helpful.

INTERFACES. What types of connections are needed? Printer, cables, RS-232, tell what they'll need to make it all work the first time.

UNIQUE FEATURES. Give password protection, and other unique features in this section.

ENVIRONMENT

EQUIPMENT. What specifically must the user have to make it run? What operating systems, and what equipment configurations will work with your package.

SUPPORT SOFTWARE. List the various utilities, and other support software required to run your program. Include the version numbers and dates of the support software to ensure maximum compatibility.

DATA BASE. If your product uses a data base, describe the nature and content of the data base. If your program develops a data base include the description of the attributes in SYSTEM DESCRIPTION.

PROGRAM MAINTENANCE

LISTINGS. Include a reference to the location of your program listings. If you need to explain the logic of the program, then do so.

PROCEDURES. Give a step-by-step procedure of how to prepare or modify the inputs, the process or outputs. Include how you would complete the maintenance task.

ERROR CONDITIONS. List the error codes, what may have caused them, and possible solutions. If you have discussed error codes in the SYSTEM DESCRIPTION, you don't need to repeat them here.

The first myth of management
is that it exists HELLER'S LAW

APPENDIX TWO
REGULATION SUMMARY

The USAF and TAC regulations in this summary impact all small computer programmers. It will help you find regulations to reference and use in your activities. Regulations change, and many installations supplement them as well, so use this section only as a guide and not your only source.

Each regulation summary is preceded by an information block. The information in the block is organized as follows:REG provides the USAF or TAC alphanumeric designation, OPR designates the office(s) the regulation effects, and C&A are agencies you can use for coordination and assistance.

REG: AFR 300-3, *Management of Small Computers*
OPR: Commander, Unit Small Computer Custodian
C&A: TAC/ADUBS & wing Data Automation managers

This regulation sets policies and procedures, and assigns responsibilities for buying, using, and managing small computers. If you currently have, or would *like to acquire* a small computer, this is the first regulation you should check out.

Written from the user's perspective, AFR 300-3 stresses practical strategies and management actions to minimize problems and get the most from small computer technology. It also defines a small computer (costs less than $25,000, operates in a stand-alone mode, etc.), and outlines your responsibilities as a user.

Policies contained in AFR 300-3 which support software development are: duplication of effort (para 10a - you **must** check with your MAJCOM before starting a project); use of BASIC as a programming language (para 10d - yes you may); documentation requirements (para 10g - should be developed as the program is built); and personal use and ownership rights of developed materials (para 12b & 12c - even if you do it on your computer but on duty, it belongs to the government). Additionally, the installation, operation accountability, maintenance, and supplies are addressed.

REG: TACR 55-57, *Use of Small Computers*
OPR: Commander, functional user
C&A: TAC/ADUBS, wing Data Automation

TACR 55-57 sets policies and procedures unique to the intelligence and operations communities concerning small computers. It must be used in conjunction with information systems regulations. This regulation outlines the process for validating and certifying software. Three types of software are described: **Standard**, **Validated**, and **User-Unique**.

Standard software is software that has been validated and certified by TAC/ADUBS. Software validation is a process that checks to see if your program will do what it is supposed to do. If you modify software that was previously validated, it must be forwarded to TAC for revalidation.

There are two categories of validated software: **safety-of-flight** and **non-safety-of-flight**. If you change non-safety-of-flight software it loses its validation and becomes **User-Unique**. User-Unique also includes software that has sufficient documentation to make it run, but hasn't been validated. TACR 55-57 describes special applications for the fighter squadrons to include: flight planning, combat mission planning, weapons delivery computation, aircrew training management, and intelligence area threat information systems.

The process for submitting your software or changing existing programs, is outlined in TACR 55-57, and this handbook.

REG: TACR 300-12, *Management of Small Computers in TAC*
OPR: Commander, functional manager
C&A: TAC/ADUBS and wing Data Automation

This regulation sets policies, responsibilities, and procedures for managing small computers in TAC. It also outlines the procedures for acquisition, maintenance, and accountability of small computer hardware and software. It applies to all organizations using or planning to use small computers.

TACR 300-12 covers many of the same policies and responsibilities as AFR 300-3, but further defines them. Use of MODEMs, unit developed software, and requirements for working with classified materials are specifically outlined.

TAC Regulation 300-12 introduces several guidelines to software development. First, you must review the TAC software catalog to avoid duplicating other programmers' efforts. Second, you must document your software in accordance with TACP 300-11. TACR 300-12 adds:

A. Software may not duplicate or conflict with other USAF or TAC standard software systems. If you get coordination and approval from HQ TAC/AD and the appropriate ADPS manager, you may duplicate the standard training, experience, and flying hour systems as a back up to the standard programs.

B. You can use BASIC, FORTRAN, COBOL, PL1, Jovial, or Pascal languages. A waiver was granted by HQ USAF for the use of BASIC for TAC unit level computer programs. The waiver for the use of Pascal is for specific applications, so check with TAC/ADUBS before using Pascal.

Appendix One of this handbook outlines the documentation requirements listed in TACP 300-11. Once written and documented, you must send your developed software to HQ TAC/ADUBS, through your local Small Computer Manager. The TAC software exchange program is outlined in Chapter Five of this handbook and details are in chapter 2 of TACR 300-12.

REG: AFR 300-10, *Computer Programming Languages*
OPR: Commander, functional user
C&A: TAC/ADUBS, wing Data Automation

AFR 300-10 sets the policy for USAF activities using or planning to use computer programming languages and programming compilers. It defines terms and outlines responsible agencies to ensure the Air Force standardizes computer programming languages. Air Force policy on structure programming is outlined in this regulation.

APPENDIX THREE
MINI-DAR FORMAT

NOTE: **A Mini-DAR should not exceed two typed pages.**

1. IDENTIFICATION: Include your functional office symbol, project officer name, and phone number.

2. FUNCTIONAL REQUIREMENT: Define the requirement in terms of *how* it will help solve your problem. Don't specify brand-names, but identify the characteristics of the requirement. If you need to process Privacy Act information, say so in the requirement statement, or your request may be delayed.

3. ALTERNATIVES CONSIDERED: Describe what happens if you don't get the software, as well as the outcome if you do.

4. RECOMMENDED ALTERNATIVE: Include the rationale for the alternative chosen. Explain how the choice will improve your capability, increase readiness, or combat capability.

5. COST/BENEFIT ANALYSIS: Show how the costs of buying software is offset by the increased accuracy, capability, speed, etc. in the request.

6. FUNDS: Identify the source of the funds. Recognize your request may very well be an unfunded requirement, and delayed for funds.

7. IMPACT OF DISAPPROVAL: State the mission impact. Express in terms of capability.

8. C_3RB REVIEW AND APPROVAL: If you are going to use any base communication cables you'll need to include an AF Form 1070 and gain approval at base level.

9. CLASSIFIED PROCESSING REQUIREMENT: Indicate if your requirement includes a need to process classified material.

10. SIGNATURE BLOCK: Your Deputy Commander for Operations and the wing SCM sign the Mini-DAR.

READER RESPONSE FORM

"There is never time to do it right,
but always time to do it over"
MESKIMIN'S LAW

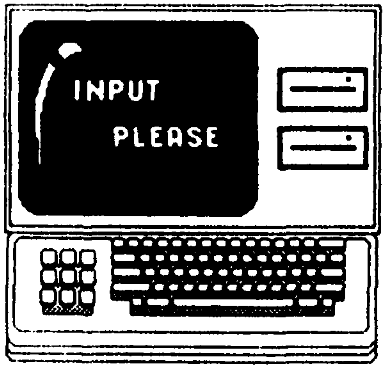
Take a few minutes, rip out this page, and sent it along with your comments to:
HQ TAC/IGIO, Langley AFB, VA 23665-5001

Was the handbook useful?

Did it help?

Would you pass it along to another user?

What changes would you like to see in this guide?



END

DATE
FILMED

9 -85

DTI[illegible]