



U.S. Army (Steve Lambert)

SOFTWARE WARFARE

FORSCOM Automated Intelligence Support System (FAISS).

The Militarization of Logic

By PETER C. EMMETT

We are witnessing an astonishing change in modern war. The volume of space in which coordinated military action takes place has greatly expanded, tending towards the global, while the time available for decisionmaking has shrunk, pushing the human operator increasingly out of the control loop. These tendencies first became apparent in air operations, but are now encompassing an increasing range of land, sea, space, and special operations,

all linked or capable of being linked digitally as never before. The vital medium of linkage is software¹ which exists in a seamless and hidden terrain: logic. Although it underpins a remarkable and growing range of military activities, software has been regarded as the plaything of engineers or an enhancement to military hardware which lacked anything of its own worthy of exploitation.

Hidden within weapons systems, and fully comprehensible only to engineers or specialists, the growing role of software in modern warfare is easy to overlook. Yet software is more than an engineering tool. It is an operational entity and weapon in its own right that needs to be exploited to maximum effect like any other. It supplants an increasing range of military functions previously undertaken by combatants. On the

Summary

Software constitutes the digital linkage among land, sea, air, space, and special operations forces, a capability that will increasingly outdistance human agents who will have to master and use it in wars of the future. It will power the flow of data, fuse information into images in command centers, analyze intelligence, and direct weapons against an enemy. Battlespace will expand as the time to make decisions contracts. The Armed Forces will rely preeminently upon near real-time adjustments to the shifting sands of war, on the ability to strike at pivotal points in small windows of opportunity. Emerging operational doctrine requires streamlined and flexible communications software that is highly dependable. Drawing on the seamless terrain of logic will enable the joint warfighter to perfect the concept of cooperative weaponry.

The views expressed in this article are the author's and do not necessarily represent those of either the Ministry of Defence or Her Majesty's Government.

ground and in the air, from beam-steering radars to intelligence-gathering platforms, software drives many increasingly sophisticated sensors with capabilities that would otherwise be limited or not available. All types of information flows via the software-driven nodes of communications networks. It is fused and transformed into images at command positions and may be further analyzed by software. Action against an enemy is conducted using weapons whose performance is also likely to be highly dependent on software.

With the power and immense potential of software as the starting point, military technology is on the threshold of a leap forward comparable with, if not greater than, revolutions that brought about the mechanization of land forces and development of airpower. The paradox and inadequacy of current thinking is this: while matériel that plays a part in war is fully militarized and exploited according to its capabilities, the potential of software is left out of the operational mainstream. The means of introducing the military functions of software into the framework of formal doctrine must now become our highest operational priority.

Earlier in this century the great task of military theorists was to reconcile war with scientific and technological innovation. That difficult task, born of the senseless slaughter of World War I, was marked by a slow acceptance of change.² Then World War II, when enemies of civilization effectively incorporated mechanization into their warfighting doctrine, brought further catastrophe to the world and near total defeat for the Allies. At the close of the 20th century, military science has an immense new challenge equally vital to the performance in battle of those Western nations to whom this still matters—the militarization of logic.

Software Military Functions

The whole art of military effectiveness lies in the ability to move cooperating forces across a theater of operations in order to strike at the decisive points, adapting as

rapidly as possible to the ever-changing and unpredictable fortunes of war. As a process this can be broken down into three basic elements: representation of the position, analysis of positional information and direction of firepower, and action of all types against the enemy. In essence, war is a cycle of “see, think, and strike” in which adaptability, intelligence, speed, and cooperation are vital ingredients. Software has now become so closely involved at all stages in this cycle that any analysis of software military functions might examine, as a reasonable starting point, the degree to which the logical analogues of each vital ingredient can be embodied within software operational doctrine.

Logical Mobility

If the analogue of firepower is processing power, then the analogue of movement in logical terrain is change—not only in the information that flows within the terrain, but also change in its logical features: the application and content of the programs fielded. Processing power designed to serve a particular tactical objective at one time may thus fail to serve it at another. To be effective, the required functionality must be adaptable. It must be logically mobile.

The tendency today is to build ever-increasing sophistication into military software. Every major and minor function gets coded, but the benefits of program sophistication are negated if it limits adapting programs in rapidly changing tactical environments. The relationship between size and adaptability is not a straightforward one. It depends on what is being changed, the number of affected program units, and the modularity of the code. The general consequences of program size must, however, be recognized and for every project its specific impact on defined functional areas must be assessed in terms of basic tactical criteria.

The Persian Gulf War resulted in emergency programming to meet unexpected challenges. These included software changes in thermal imaging and laser designator pods installed on aircraft by the Royal Air Force³ and Firefinder weapon-locating radar adapted to a Scud missile detection role by the U.S. Army.⁴ Such ad hoc program changes, implemented in time to be put to tactical use, represent the beginnings of what will undoubtedly be an important facet

Squadron Leader Peter C. Emmett, RAF, is posted to the Defence Research Agency, Great Malvern. Previously he was assigned to the Aircraft and Armament Experimental Establishment, Boscombe Down.

of future software operational doctrine, namely, tactical programming. This may be defined as transforming assets in logical terrain to deal effectively with the challenges of the moment. As software increasingly penetrates the tactical level, so will the need to exploit the adaptability characteristic of software. A compromise will be required between program sophistication, on one hand, and adaptability, on the other. Such a compromise is comparable with that between armored firepower and mobility. In this critical tradeoff software engineering is failing utterly to serve real military needs.

as software penetrates the tactical level, so will the need to exploit the adaptability of software

In the early days of computer programming the bulk and limited capacity of available digital storage technologies were severe constraints on program design and, hence, on the sophistication of system requirements. It also led programmers to find the slickest means of horning the required functionality into the available memory space, and favored efficient but highly obscure languages and programming techniques. Such programs were difficult to comprehend, but at least they were bounded and the implemented functionality was reduced to the operationally essential. With the exponential growth in memory capacity from the 1960s onwards, program size simply grew to fill available memory space. With diverse programming languages and poor development techniques, the need arose for programs in standard high-level languages in a framework of agreed software engineering discipline.⁵ And as software engineering leapt ahead leaving operational staffs in a void of subservient incomprehension, it effectively hijacked the procurement process.

The quantity of software generated for military equipment today staggers the imagination. Freed from the bounds of computer memory size and aided by a growing plethora of development tools, programming teams routinely churn out lines of code by the millions or tens of millions. At the outset of a project, operational staffs are beguiled into building every conceivable sophistication into a project requirement, supported by a seemingly limitless capacity of

engineers to generate the required code. In too many cases they are oblivious to development risks, maintenance costs, and program adaptability. The result is that the terrain of modern combat is filled by non-adapting dinosaurs—monsters of functionality bloated by excess requirements, the essential along with the unfiltered trivial. Such weapons will have no place on the logical battlefield of the future.

The latest rash of problems with the new generation of software-intensive fighter aircraft⁶ should give pause for thought on the achievements of software engineering in the cold light of the military balance sheet. It is not that software cannot bring immense new capabilities. Rather, it is a question of placing the untamed power of software within a doctrinal harness to obtain effective military benefit from the capabilities. A primary task of doctrine is thus to impose a strict review process on the military functions that ought to be trusted to code and on the total extent of code that may be generated. The appropriate maxim might be: If it doesn't win wars, don't code it. The central aim must be to strike the right balance between war-fighting capabilities and future demands to alter those capabilities when the need arises.

The ideal adaption is one that can be generated in real-time during the course of battle. This is the domain of Artificial Intelligence (AI) that can be regarded as the ultimate in logical mobility. Though it is unlikely that software adaption can ever be uniquely of this form, software operational doctrine must embrace AI as a key strategic technology.

Battlefield Real-Time

The speed and complexity of modern conflict are leading inexorably to trust in a growing range of functions in the "see, think, and strike" loop to automated actions governed by software. A simple example is the automatic fire mode of guided weaponry, such as the Patriot anti-missile system, in which the linkage among sensor, threat analysis, and fire decision (with optional human override) is fully computerized. On the wider battlefield, if command and control decisions are to be made in tactically meaningful time, the sophistication of modern sensors and the vast quantities of



U.S. Air Force (David McLeod)

H-64 Apache assault helicopter.

software-intensive weapons generate the need for speed in threat assessment

information they generate means that the battlefield analysis process must itself be increasingly entrusted to software. An immense research effort in this area is already generating the tools of the future, such as the “Warbreaker” data base for finding time-critical targets being developed by the Joint Intelligence Development Staff.⁷ Software-intensive weapons thus generate the need for speed in threat assessment and, as tactical activity in war speeds up, the requirement for speed of decision generates the further need to trust intelligent systems. The human element will always be present, but it is being progressively swamped, marginalized, and obliged to depend on capabilities and flexibilities written into command and control software at the outset of a conflict.

Preparation and innate adaptability, achieved by sound doctrine, are the keys to future combat effectiveness.

The greatest changes in the practice of warfare are likely yet to come. On the battlefields of the present era, software is omnipresent but exists in each case as a servant of some well-contained command and control function. Intercommunication among software elements may take place, but the human combatant remains the principal means of linking the “see, think, and strike” loop. It is important to recognize the strong

temporal dependency of any threat and that the more effectively an opponent operates in logical terrain, the faster the threat will change. The human link becomes ever more the weakest, and inexorably automatic fire modes will replace slower manual processes as this becomes technically feasible. However, this cannot be a localized development confined to weapon software in isolation from other elements on the battlefield. In order to exploit the speed characteristic of software, there must be direct linkage and control back to the command and control position. Only here can the full picture be assessed and the most effective strike modes be identified.

The central aim is, as always, to coordinate diverse forces to strike most effectively at the decisive points *at moments of vulnerability*.

With many competing demands for information and direction, a mechanism for resolving conflicts and allocating priorities will be essential. The analogy with the problem of real-time control in a multi-tasking engineering environment—well understood by the software engineer—is both striking and perfect. The logical battlefield is steadily evolving into a single, massive, real-time system in which human activity represents a subset of the total process. What is presently lacking is a battlefield real-time executive that is able to resolve conflicts and allocate operational priorities between competing and ever-changing demands on military hardware use. This resource must be able to employ the results of sensing and analysis to generate continually the optimum instantaneous strike posture for each weapon. Weapons would be assigned singly or by group and switched between local and autonomous control as necessary. The object would be to direct and coordinate available firepower at the weakest areas *as soon as the moments of opportunity arose*. The rationale of battle would be embedded in a



U.S. Army (Steve Lambert)

Soldier Integrated Protective Ensemble (SIPE) with image intensification and thermal sight, laser aiming light, and Load Bearing Component for computer or microclimate conditioning.

cooperation among software-driven weapons is a potent tactical concept

suite of analysis programs on continual call to the battlefield executive. An underlying operational doctrine must be wedded to battlefield intelligence here in order to generate specific operational directives. Here as well, operational flexibility must be maximized through program adaptability. The battlefield executive system would be nested in higher order executive programs. At the highest executive

level would reside the strategic rationale from which immediate tactical priorities would be derived.

The exploitation of the software characteristics of speed, intelligence, and adaptability can be traced in the future battlefield system alluded to above. In the strict sense that the system would be centrally coordinated, a type of logical cooperation would exist. But combatants can also cooperate without recourse to higher command, an activity with no logical analogue as yet.

Logical Cooperation

Cooperation among software-driven weapons is a potent tactical concept. It is the artillery on the logical battlefield just waiting to be discovered. Totally overlooked in old ideas about war, it is the most compelling evidence of the need to alter perceptions of the nature of modern warfare.

The capabilities and potential of cooperative weaponry can be illustrated by the example of a stand-off tactical air-to-ground missile. Its requirements and specifications must be considered in terms of defined targets, for no single design could ever cope with the myriad of possible ground targets. For instance, concrete bunkers may be identified as the principal targets. A requirement to penetrate a given thickness of concrete will then be specified and the missile's body and warhead designed accordingly. Such a weapon will be

less effective in roles outside the design parameters, such as area destruction of buildings or blast attack against scattered ground targets. By contrast consider the many possible modes of operation for a group of cooperating air-to-ground missiles. Under the control of a coordinating software, resource options would be open to attack in sequence or simultaneously and at one location or many. The results of an attack could also be employed in selecting targets for follow-on attacks, the selections being made instantaneously (real-time) by the directing software. Simultaneous or closely sequenced attacks on a single location would tend to be effective against hardened targets, while sequenced strikes against buildings could be employed until a desired effect was achieved. The many possible combinations could be selected either on a self-organizing basis or in response to directives from a central command and control position.

Weapons would attack in groups and act as cohesive entities, adapting to the characteristics of any threat and wrapping themselves around it at the weakest points. The significant benefits of logical cooperation would be greatly improved flexibility in weapon design and use, much greater collective destructive effect, and a potentially lower detection threshold achieved through the ability to disperse in defense and swarm during attack. A low detection threshold would favor developing small weapons. The joint applications of logical cooperation are as diverse as war itself, from air combat to undersea mine warfare. Its exploitation is likely when a need arises to apply force with maximum economy at times and places that will make an operational difference.

The Operational-Engineering Relationship

The role of doctrine is to serve as a guide to action, but software operational doctrine raises special problems of application because of the sheer complexity and esoteric nature of software. To be applied effectively, the relationship between the operational and engineering domains of military software development has to be clear.

Unbounded software production has led to project overruns, escalating costs, and

products that fail to meet design specifications. These widely recognized problems have been described as the “software crisis,” and solutions have repeatedly been sought using a software engineering approach. However, the role of the engineer is to deliver on a request, not to determine its nature and scale. The crisis stems not uniquely from bad engineering but ultimately from inappropriate and anachronistic procurement philosophies that have failed to adapt to the peculiar difficulties of software development. In the military sphere, the way out of the crisis must be sought through operational doctrine that, by identifying the military functions of software, harnesses and directs the power of software within a coherent procurement rationale. Software operational doctrine will take years to establish but its relationship with software engineering may be simply stated. The purpose of software engineering is to maintain the operational capacity to deliver and support the new or altered functionality requested by the user, *in accordance with operationally*

defined criteria. The role of software operational doctrine is to determine the scale, content, and operational quality of what ought to be requested. Thus this doctrine must not only be imposed firmly upon software engineering at all stages of project development, but it must also unhesitatingly interfere in

the methodologies employed in software engineering. Software production is an operational issue.

As an example of the strategic insight that may be obtained through software operational doctrine, consider the need to establish, through doctrine, a set of criteria for operational quality. Operational quality has many facets but includes the frequent need to trust software to carry out its functions reliably. Degrees of required reliability are usually expressed in terms of the safety or mission criticality of program functions.⁸ Such

classifications are appropriate when programs undertake well defined and compartmentalized support roles in a weapon or system and thus can be segregated effectively from less critical software. But as the role of operational software widens and is utilized more as a weapon in its own right, forging multiple and fast-changing linkages in fluid tactical environments, the distinctions between safety critical and non-safety critical software will become increasingly meaningless. Given the unpredictability of war, the importance of any single unit of code cannot be predetermined. This leads to the need to treat all operational software on the same basis and to the concomitant demand for performance reliability that is as near absolute as possible. Only mathematical proof can deliver such levels of assurance.

Processes by which programs may be demonstrated to be correct by mathematical analysis are termed Formal Methods (FM). These methods take many forms, but an important one is the capture of informally conceived requirements as a formal, mathematically describable specification. This is an intensive manual process that now can only be done by a limited number of specialists. Once a formal specification has been defined, however, the code generated may be analyzed with near absolute rigor, and a high level of trust may be placed on program performance.

Defining a formal specification has another potential benefit that future military leaders must fully grasp: it is a major step towards automatic code generation.⁹ It may take a strong measure of operational awareness to advance the science of FM, just as armored warfare led to advances in metallurgy and mechanics, but achieving automatic code generation would represent a strategic asset of the highest order. Widespread use of FM and automatic code generation would lead to a major shift in the manual effort of programming (still required in hardware interfaces) towards the equally manual process of formal requirements capture. The major strategic gain would be one of greatly enhanced logical mobility through the ability to convert rapidly an informally conceived but immediate tactical need in a reliable battlefield program.



U.S. Army

Flight simulator showing Forward-Looking Infrared Radar (FLIR) image from Target Acquisition Designation System (TADS).

Automatic code generation would not, however, mean reducing the personnel requirement for generating military code and may well increase it. What it would do is alter radically the pattern of personnel use, both in operations and support. Operationally, the two principal roles would be control over operational program configuration and the continuous definition of informal software requirements. In the support area requirements would be captured formally by a team of dedicated specialists and converted into reliable code for immediate use in the theater of operations.

The organizational expression of the relationship between software engineer and combatant must also be a result of software operational doctrine, born of the need to ensure fast and effective cooperation between

winning wars must be the overriding criterion of all software development

the support and operational areas of software combat. Viewed in this way the software engineer-combatant relationship may be compared with the well established relationship between field service teams and armored combat units. Both are concerned with the maintenance of field mobility, whether physical or logical.

The Future Battlefield

That software is a weapon in its own right is the justification and foundation of software operational doctrine. Winning wars must be the overriding criterion of all software development. There is no room for excess baggage in software warfare. The lumbering monsters of code on the nascent logical battlefields of today must give way to lean, adaptable, and communicative software that undertakes well-defined functions in an all-embracing strategic system.

Starting with the military prerequisites of adaptability, intelligence, speed, and cooperation, software operational doctrine has been considered in terms of logical analogues. Derived concepts of logical mobility, battlefield real-time, and logical cooperation have also been identified as a basis for further doctrinal development. From doctrine to practice, through a clearly defined operational-engineering relationship, arise requirements for speed in software development and absolute trust in operational software.

The idea of cooperative weaponry stems from considering software as a weapon in its own right, as opposed to a tool of particular hardware performance requirements. The technology for putting logical cooperation into practice has been available for many years. The missing ingredient has been systematic thinking to illuminate the strange new terrain on which the military of the next century will undoubtedly operate. The concept of cooperative weaponry surprised many when it was introduced in 1992. As the illumination of this new terrain increases it is safe to assume that logical cooperation will not be the last surprise to be found. JFQ

NOTES

NOTES

¹ Peter C. Emmett, "Software Warfare: The Emerging Future," *The RUSI Journal*, vol. 137, no. 6 (December 1992), p. 56.

² Robert H. Larson, *The British Army and the Theory of Armored Warfare, 1918-1940* (Newark: University of Delaware Press, 1984).

³ P. Jackson, "TIALD Designated a Success," *Royal Air Force Yearbook Special* (1991), p. 33.

⁴ "Trashing Iraqi Scud Sites Shows Firefinder's Mettle," *Signal*, vol. 45, no. 12 (August 1991), p. 27.

⁵ Barry W. Boehm, *Software Engineering Economics* (Englewood Cliffs, N.J.: Prentice-Hall, 1981).

⁶ D. Learmont, "Report Challenges Gripen Controls," *Flight International* (October 25-31, 1993), p. 5; Charles Bickers, "Critical Paths: Who's to Blame When the Technology Fails?" *Jane's Defence Weekly*, vol. 20, no. 19 (November 6, 1993), p. 24; and A. Jeziorski, "Eurofighter First Flight Pushed Into Next Year," *Flight International* (October 13-19, 1993), p. 4.

⁷ R. Ropelewski, "Team Helps Analysts Cope with Data Flood," *Signal*, vol. 47, no. 12 (August 1993), p. 42.

⁸ United Kingdom, Ministry of Defence, "The Procurement of Safety-Critical Software—Defence Equipment" (Interim Defence Standard 00-55, Part 1: Requirements, Issue 1.5, April 1991); and RTCA/EUROCAE, "Software Considerations in Airborne Systems and Equipment Certification" (DO-178B, 1993).

⁹ See F.L. Bauer, "The Wide Spectrum Language CIP-L," Munich project CIP, *Computer Science*, vol. 1 (1985), p. 183; and K.C. Lano, "Validation Through Refinement and Execution of Specifications" (REDO project document 2487-TN-PRG-1041, August 1990).

READERS who want to share their thoughts on "Software Warfare" with the author may write to him directly at:

Defence Research Agency,
St. Andrew's Road,
Great Malvern,
Worcestershire WR14 3PS
U.K.