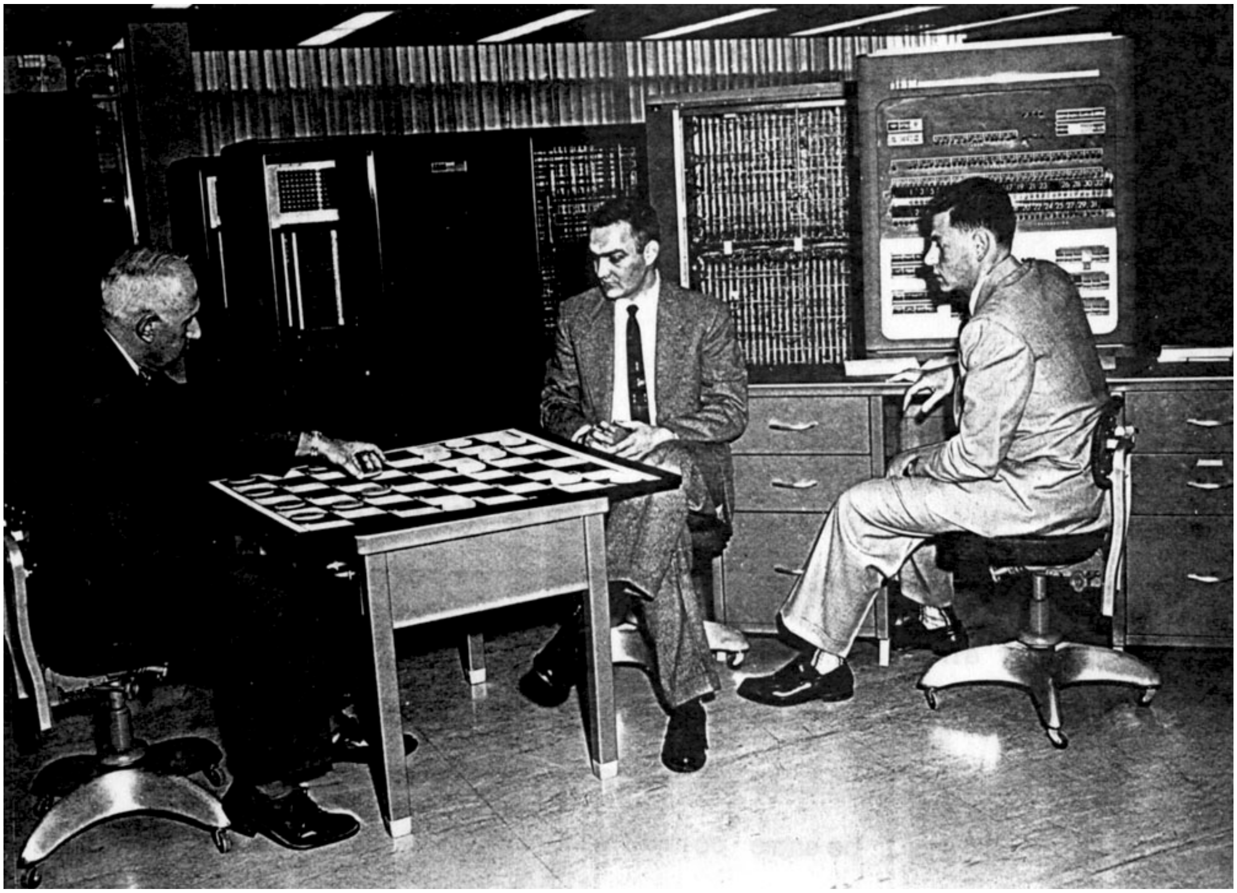
A Marriage of Convenience:

THE FOUNDING OF THE MIT ARTIFICIAL INTELLIGENCE LABORATORY



Stefanie Chiou
Craig Music
Kara Sprague
Rebekah Wahba

A Marriage of Convenience:

THE FOUNDING OF THE MIT ARTIFICIAL INTELLIGENCE LABORATORY

Stefanie Chiou
Craig Music
Kara Sprague
Rebekah Wahba

ABSTRACT

The MIT Artificial Intelligence Laboratory began in 1959 as a small group of research students, systems engineers and interested faculty. In 1963 the group joined Project MAC, an effort to bring together research groups from all over the Institute that were interested in computing. From the very beginning of Project MAC, the AI Group represented a significant portion of the project, receiving roughly one third of its total funding from ARPA. However, due to a number of reasons including differences in mission and culture, the AI Group and the rest of Project MAC never became bound to each other. By 1970, they depended on each other only minimally and they were not relevant enough to each other to necessitate being part of a single entity. The 1970 decision of the AI Group to split from Project MAC was due to a confluence of issues and only formalized a split that had already existed between the two groups for a considerable time.

6.933J/STS.420J
Structure of Engineering Revolutions
Project History Rough Draft
Professor David Mindell
TA: Chen-Pang Yeang
December 5, 2001

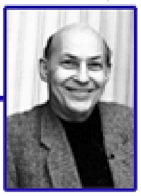
Table Of Contents

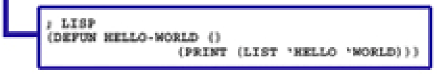
0. TIMELINE

1956	*Dartmouth Summer Research on AI*
1957	*Sputnik* *704 donated to Computation Center by IBM*
1958	*ARPA created* *MIT AI Group founded by McCarthy and Minsky* *LISP implementation begins* *TX-0 given to MIT by Lincoln Labs*
1959	*McCarthy's memo on timesharing*
1960	*Long Range Computer Study Group*
1961	*PDP-1 given to MIT by DEC* *Corbato debuts CTSS*
1962	*McCarthy leaves MIT for Stanford*
1963	*Project MAC is started*
1965	*Multics development begins*
1967	*ITS Development begins* *Multics initial implementation*
1969	*ITS is introduced*
1970	*AI Group leaves Project MAC to start AI Lab* *Minsky wins Turing Award and gives Turing Lecture*

J. McCarthy

M. Minsky





```
; LISP
(DEFUN HELLO-WORLD ()
                (PRINT (LIST 'HELLO 'WORLD)))
```

F. Corbato

1. Introduction

The MIT Artificial Intelligence Laboratory has existed, in one form or another, since 1959. The lab was founded on the principle that vision, robotics, and language are the keys to understanding intelligence, and ultimately how the human mind works. In some ways, AI was one of the very first computer science rigorously studied at MIT, and thus pioneered many of the computing practices that are now standard in the field of computer science.

The story of the AI Lab goes back to the days of mainframe computers, which were so expensive that they were usually shared among a number of unrelated groups. As the demand for computational power increased throughout the late 1950's and early 1960's, MIT's solution to the rising demand for computer time was to implement a computer usage model called "batch processing." Batch processing prevented user interaction with the computer for those who were actually using its processing power. It also significantly lengthened the iterative process by which most programs developed, from implementation to debugging and application.

This environment of mediated and slow computing was frustrating and an enormous obstacle to emerging research in Artificial Intelligence, which had only formally began at MIT in 1959. Timesharing was conceived of as a means for the AI researchers to interactively debug their programs from a terminal attached to the large mainframe at the same time that the large computer was also engaged in its normal batch processing. In the years that followed the initial introduction of timesharing, a number of systems were developed to enable computers to multitask. In 1963, Project MAC was an attempt to bring all of the computer researchers at MIT together into a single community. The project had a number of objectives, including the implementation of a timesharing system that would bring computing to the masses, and research into machine-aided cognition that characterized the work of the AI Group.

While the AI Group was intended to be a significant part of this larger computing-oriented body, it never became an integral part of Project MAC. Initial differences in the management of the two organizations, the computing culture by which they were characterized, and their respective missions never seemed to disappear. At the same time, infrastructural issues, such as space, funding, and the computing environment, were increasingly stressing the tenuous ties of the AI Group to the rest of the project. Finally, in 1970, after a seven-year marriage to Project MAC, the AI Group left the project to form its own laboratory, now called the MIT Artificial Intelligence Laboratory.

Perhaps the most interesting fact about the separation of the AI Lab from the rest of Project MAC is that there was no single event or issue that "broke the camel's back" and directly prompted the split. Rather, the split was brought about by a confluence of issues that were a combination of pre-existing differences in culture and vision; and a culmination of more mundane matters that simply made separation a more valuable alternative to staying together. The history illuminates the reasons behind why the AI Group's seven-year marriage to Project MAC ended not with a bang, but with a whimper.

This paper presents a history of the founding of the MIT Artificial Intelligence Laboratory, starting with its roots in early computing and the fledgling efforts to understand how the mind works. It examines the emergence of timesharing and the widespread implications that this new

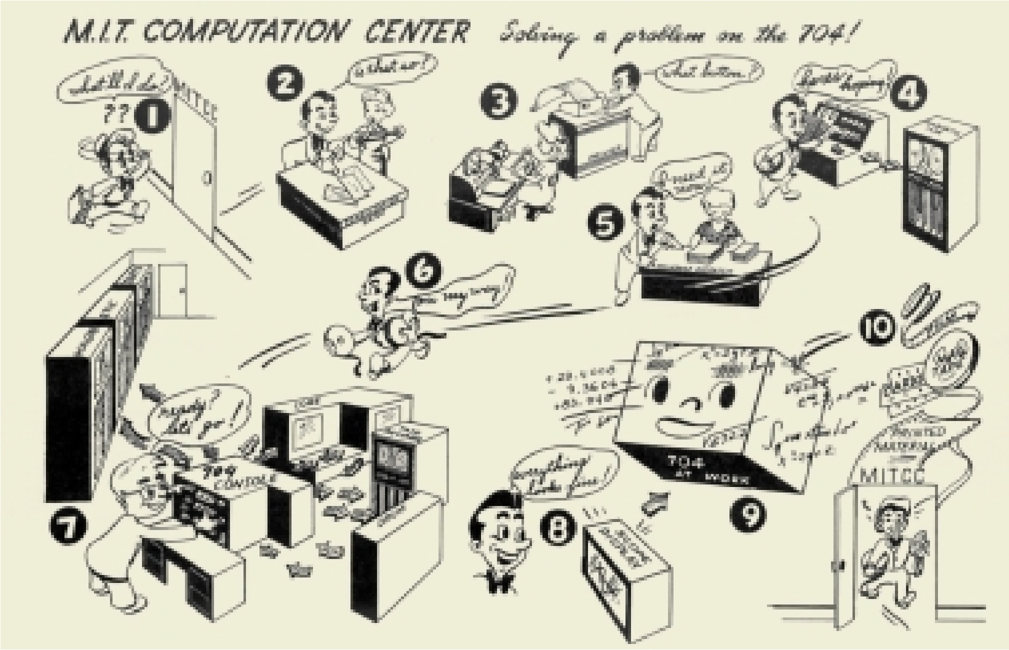


Figure 1: Comic about using the IBM 704 in the MIT Computation Center

computing technology had on the computing environment at MIT. It further discusses how the emerging AI Group formed the basis for a dedicated research laboratory in computer science at MIT, yet somehow resulted in the alienation of the very group that was initially so dedicated to the concept. Interwoven throughout are issues surrounding the computing culture of the 1950's and 1960's and a description of how the trajectory of man-machine interaction was shaped by a number of forces, both technological and social. The story of the founding of the AI Laboratory and the development of timesharing also demonstrates a number of properties that are common to engineering revolutions throughout history.

2. EARLY COMPUTING AT MIT

Computers of the 1950's were bulky, unfriendly machines that required large rooms filled with air conditioners and cooling equipment to ensure that they would not overheat. In addition to needing a great deal of space and energy, these computers also required a substantial technical staff to run and maintain them. The users and programmers of the computers did not really interact with the machines on an individual level. Instead, a series of machine operators and technicians mediated all interaction since they were the only ones who could actually touch and manipulate the machine. Users of the machine could only present a set of program punch cards and data punch cards to the machine operators, who would then feed the program cards and then the data cards into the machine. The machine ran the program on the data input and output the results to another set of punch cards, which the user received at the end of the computation. Since computers in the 1950's were so expensive, there was no better model for computing that would afford users more interaction with the machines. For example, the IBM 709 cost $3 million when it first debuted in 1957[1], which was impossible for a single person or even a small research group to afford. Instead, large groups of people shared the computer in order to bring down the overall cost of ownership and operation. The sharing was accomplished by dividing the computer's time into shifts; the computer had a certain amount of time to process a problem for one group or person until their shift ran out at which point the computer would be used to process another problem for a different user.

2.1 Batch Processing In the Computation Center

For a large part of the 1950's and the early 1960's the MIT Computation Center, housed on the ground floor of Building 26, was responsible for serving the computing needs of the Institute. During this time, the Computation Center was under the Directorship of Philip Morse. John McCarthy and Fernando Corbato were both faculty associates of the Computation Center. In 1957, IBM donated a 704 computer to the Computation Center. The donation came with the stipulation that the computer's time would be divided into three eight-hour shifts, one of which would be given to MIT, another to a number of universities and institutions along the East Coast, and a third to be devoted to processing for IBM.

Figure 2: Batch processing delays

Despite the generous donation, the computing power provided by the IBM machine simply was not enough to meet the demands. MIT's response to the increasing demand for computational power was twofold. The first measure was purely technological and involved upgrading the system whenever possible. By 1963, the Computation Center's machine had evolved from a 704 to a 709 to a 7090 (the transistorized version of the 709), and finally to a 7094. Despite the consistent upgrades, though, the Computation Center was still unable to meet the increasing demands on the system.

The second measure that MIT employed was to implement a computer use model called batch processing. Batch processing was a method by which the operators of the machine would collect all of the programs and data of the day and punch them, in order, onto a tape. The tape was then run through the machine during the appropriate time shift. When a "batch" of programs was finished, the users could return to the Computation Center and pick up the results of their computation. Batch processing did help to manage the resources of the computer more efficiently, but it resulted in longer delays, on average, from when the user first dropped off his

[1] http://www.multicians.org/history.html

program to when he could get back the results. Another cost of the batch processing system was that it further removed the programmer from interacting with the machine.

2.2 Interactive Computing In The Research Laboratory For Electronics

The batch processing system used in the Computation Center can be contrasted to a more interactive approach to computing that found its roots in the MIT Research Laboratory for Electronics (RLE), which was located on the second floor of Building 26 just above the Computation Center. Whereas the Computation Center received its machines through donations or purchases from IBM, RLE obtained old machines from MIT's Lincoln Laboratory to use for its own computational purposes. In 1958, the Lincoln Labs gave a TX-0 computer to RLE on a "long-term loan"[2]. The TX-0 was originally developed to test the new magnetic core memory of another computer called the TX-2, which was the first transistorized computer. After completing the testing of the memory, Lincoln Labs no longer had any use for the machine and thus donated it to RLE where it was put under the supervision of Jack Dennis.

The TX-0 was the first machine that included a terminal at which the programmer could actually sit. The computer also abandoned the use of punch cards in favor of a system in which users punched their program onto a long, thin paper tape with a Flexowriter typewriter. The users would then sit at the console and feed in the program by running the tape through a reader. The most exciting part of the TX-0 system was that users were able to sit there and watch as the program ran. Furthermore, if something went wrong, the programmer would know immediately and he could debug the problem by using the levers and handles on the machine. Dennis himself liked to write code and since the TX-0 had been stripped before it arrived at RLE; it had almost no software on it. One of the first programs Dennis wrote was an assembler for the machine. Then he introduced the concept of a debugger, which allowed programmers to find bugs in a program while they were at the terminal, fix them and keep the program running. Dennis called this first debugger for the TX-0 FLIT, which stood for Flexowriter Interrogation Tape.

The TX-0 model of computer interaction, in which a single user had full control of a moderately interactive computer, was to leave an indelible print on those who became intimately acquainted with the machine. Since a programmer could actually interact directly with the computer, an arrangement was needed in order to allow everyone to have a fair chance to use the machine. At the time, the sharing arrangement developed for the TX-0 was very similar to that used in batch processing where machine time was shared between different groups of users. With the TX-0, users were given shifts of time in which they would have individual access to the computer. Although the computational resources were still limited, the TX-0 was also much more accessible to the MIT community than the hulking IBM machines. This availability was soon discovered and exploited by the curious students of the Tech Model Railroad Club (TMRC).

2.3 The Tech Model Railroad Club

The Tech Model Railroad Club was a student-run organization with headquarters in the basement of the former Building 20. While it was singular in purpose, the group had two very different factions who frequented their clubhouse. On the one hand, there were those students who loved spending their time painting replicas of historic trains. On the other hand there were

[2] Steven Levy, Hackers. (New York: Penguin Books, 1994). p.30.

those inhabitants of “the notch”[3]; a small back room where the control system for the tracks and railways was housed. These students were far more interested in the machines and systems that controlled the tracks than they ever were in actual trains. They were so keen on learning how every single switch and handle affected the control of the trains on the track that they would spend hours and hours improving and revamping the control system. They also developed a habit of rummaging around campus in the middle of the night, looking for broken or unwanted computer and telephone parts that they might be able to use and add to their railroad control system. These students of the TMRC were the very first hackers at MIT.

These hackers demanded that their interaction with machine be immediate and even visceral. They were disciples of the Hands-On Imperative, which meant that they hated the mediated interaction that surrounded hulking IBM machines. There was a “Priesthood”[4] of bureaucrats and pseudo-technocrats who prevented them from truly accessing and exploring the workings of the machine. Thus, when the TX-0 arrived at RLE, the hackers jumped at the chance to actually sit down and interact with the machine on a real-time basis.

Figure 3: Tech Model Railroad Club members

Jack Dennis was a benevolent father-type to these young hackers, some of whom were still in high school. He fostered an atmosphere in the lab where it did not matter what project was worked on, just that people were learning how to manipulate the computer. The hackers who found their way to RLE were also introduced to Marvin Minsky, a faculty associate with a keen interest in both computers and the budding computer science field of Artificial Intelligence. The hackers soon became accustomed to the singular-ownership of, and interactive relationship with, the computer. These values would persist in hacker culture for many years to come.

2.4 The MIT AI Group

Having both been involved in the Dartmouth Summer Research Project on Artificial Intelligence that McCarthy had organized in 1956, John McCarthy and Marvin Minsky had been acquaintances and colleagues in the study of Artificial Intelligence even before they both joined the faculty at MIT in 1957. In 1959 the two united as co-Directors of the MIT AI Group:

> They were walking the halls of building 26, which was then RLE... and they saw the Director and said, we would like you to fund a group dealing with Artificial Intelligence... and he said, OK. And then he said, how much do you need, and they said, $50,000, which was a lot of money in those days, this was about 1958, and he said, OK. And that was it. That's how they started. [5]

[3] Ibid. p.21.
[4] Ibid. p.19.
[5] Joel Moses. Interview conducted November 29, 2001.

The AI Group immediately focused their research efforts on expanding LISP (LISt Processing), a programming language first introduced by McCarthy in 1958, and applying it to problems of Artificial Intelligence. The original purpose of LISP was to study a list of formal declarative and imperative sentences in such a way as to make deductions from them.[6] In the following years, a number of technical memos were published on topics ranging from puzzle solving and calculus in LISP to computer chess players. Though McCarthy eventually left MIT in 1962 to start his own AI lab at Stanford, his departure was quickly followed by the arrival of Seymour Papert, who helped Minsky direct the AI Group for the next decade.

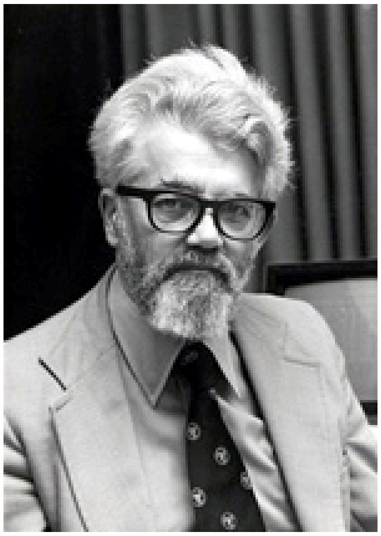


Figure 4: John McCarthy

From the outset, the AI Group was composed of a number of very bright, outside-the-box thinkers. The students who originally joined McCarthy and Minsky were of a special breed: "when those first students came to work with us, they had to bring along a special kind of courage and vision, because most authorities did not believe AI was possible at all."[7] The group's membership included "youngsters [that] came to MIT from all over the world, obsessed and inspired as much by science fiction fantasy as scientific papers – as much by Asimov, Heilein and Pohl as by Turing, Shannon, and McCulloch."[8] The strong contingent of students exclusively interested in Artificial Intelligence problems was complemented by a number of hackers from the TMRC that were attracted by the computers that Minsky had acquired for his group and encouraged by the open environment that he fostered. Minsky "knew that to do what he wanted, he would need programming geniuses as his foot soldiers – so he encouraged hackerism in any way he could."[9] After witnessing the convergence of the TMRC on the TX-0 and later the PDP-1 of RLE, Minsky learned how to attract the hackers he needed for his group:

> Marvin Minsky decided that what he wanted to do was encourage very smart people to play and he bought a lot of toys and opened it up and everybody who wanted to play could come play with his toys. The people who liked toys, especially toys which were very complicated and full of controllable parts showed up, and those were the same people from the Tech Model Railroad Club.[10]

3. TIMESHARING

John McCarthy is credited with first conceiving the notion of timesharing, though the body of literature suggests a number of influences and parallel developments. His work on LISP resulted in a great deal of contact with the Computation Center's 704 computer and batch processing

[6] Karl L Wildes, and Nilo A. Lindgren. A Century of Electrical Engineering and Computer Science at MIT 1882-1982. (Cambridge: MIT Press, 1985) p. 267.
[7] Levy, p. 62.
[8] Ibid. p. 62.
[9] Ibid. p. 67.
[10] Gerald Sussman. Interview conducted November 15 and 29, 2001.

system. His frustrations with the status quo centered primarily around the lack of interactive debugging that was available on the batch processing computers. Saltzer explained that “in order to do an AI application you need to be able to debug programs. And this business of batch processing was death on AI programming. Because you need to be able to have a dozen interactions an hour rather than one a day. So they needed an interactive system in order to do AI research.”[11] In a memo to his superior, Philip Morse, dated January 1, 1959, McCarthy explained the difficulties he was experiencing trying to program using the traditional batch processed computers. He suggested that the only way that real-time interactive debugging would be achievable at a bearable cost was by timesharing the machine. In his vision of timesharing, the computer can attend to other customers while a customer at the terminal is reacting to some output.

The faculty and staff of the Computation Center enthusiastically embraced McCarthy’s vision of timesharing, as they shared his frustration with batch processing. Computers that could service more than one user at the same time could help them to meet the saturated demand for the processing power of their only mainframe computer. Herbert Teager, an Assistant Professor in Electrical Engineering, was put in charge of timesharing development in the Computation Center. His team succeeded in connecting an electric typewriter to the IBM 704 and producing a programming system that allowed the 704 to be timeshared. Following this initial success, Teager set out to build an even better timesharing system for the 704. His efforts, which were seen as overly ambitious by many of his peers, were eventually taken over by Fernando Corbato, then the Associate Director of the Computing Center.

In addition to early work in timesharing at Bolt, Baranek, and Newman (BBN), a local Cambridge company with strong MIT connections, at least two timesharing system projects began at MIT in 1960. One effort was lead by Corbato in the MIT Computation Center for the IBM 704. His objective was “to be the computer service for the masses. The important thing was that everybody would be there and that they would have confidence that their files would be there tomorrow.”[12] Admittedly, his initial effort, what later was called the Compatible Timesharing System (CTSS), was more of a hack aimed at demonstrating the value of timesharing systems to the rest of the world. Corbato had been personally frustrated with the system of batch processing, which tied up the whole system for a single user. He wanted to allow people access to the machines, as opposed to submitting punch cards to some intermediary, while still preventing any single user from bringing down the whole system. The computing model that came out of the MIT Computation Center’s research into timesharing was that of the “computer utility.” The notion of the “computer utility” was that the computer was akin to an electric utility – anyone who wanted to could plug into an outlet, a terminal in this case, and take out what they needed. Just as the electric utility isolates the different users through a series of fuses and switches, users of the computer utility were also insulated from everyone else.

A second timesharing effort began in the MIT Research Laboratory for Electronics in 1960. The Digital Equipment Corporation, a computer manufacturer started by ex-members of the TMRC, had donated a PDP-1 computer to MIT to be installed in the same room as the TX-0 experimental transistorized computer. Jack Dennis managed the new machine and, with the help

[11] Jerome Saltzer. Interview conducted November 19, 2001.
[12] Jack Dennis. Interview conducted November 29, 2001.

of his students, implemented timesharing on the new computer so that they could still use the computer while it controlled other equipment in the laboratory. Where Corbato's system aimed to bring computing services to the masses, using the idea of the "computer utility", Dennis and his students wanted a system that could control and monitor the equipment in the laboratory while still providing the best experience to the user as possible. User programs could interact with laboratory equipment while at the same time, interactively servicing other users' requests. Dennis remembers that they had the machine "fixed up so a radio astronomists' group could run a program to control an antenna on the roof at the same time that other people could do what they needed to do on the machine."[13] The PDP-1 timesharing system also included the micro-FLIT debugger, an early predecessor to the DDT debugger later developed by the AI hackers. He had stripped off much of the features of his earlier interactive debugger to make the program fit on the PDP-1, so micro-FLIT was simply a reduced version of the FLIT debugger running on the TX-0. The RLE timesharing model was later to be an enormous influence on the computer systems designed by the hackers of the AI Group.

Figure 5: Jack Dennis

Just as these timesharing efforts were getting under way, MIT began to plan for its next generation computer system. Demand for use of the Computation Center's 709 had already exceeded the capabilities of the system and the Institute was feeling more and more internal pressure to figure out how to provide more computational power to its own community. Between 1960 and 1961, MIT sponsored a committee called the Long Range Computer Study Group to examine the future of computing resources at MIT. The high-level supervisory committee consisted of Philip Morse, Albert Hill, and Robert Fano, another faculty member from RLE and an advisor to McCarthy and Minsky. Herbert Teager headed a lower-level committee including Dennis, McCarthy, Corbato, and a few others. McCarthy eventually took over as chair of the committee following a series of disagreements with Teager about his ambitious and often vague plans. McCarthy's committee submitted a report suggesting that MIT take steps to acquire a large timesharing computer to provide the community with computing services. MIT initially failed to take action on the report partly because the cost estimates were very large, and partly because IBM had promised to meet MIT's needs at little or no cost if MIT would just wait until the IBM 360 computer was completed. Unfortunately, the design of the 360 fell behind schedule. IBM and MIT relations also began to falter due to a patent lawsuit over the invention of magnetic core memory. As part of the stalling effort, President Stratton suggested a new study to rigorously establish the demand for timesharing among the MIT computer users. McCarthy was not at all interested in such an undertaking and decided to take the opportunity offered to him by the president of Stanford University to return to California and build a Computer Science Department at Stanford.

[13] Ibid.

4. PROJECT MAC

MIT stalled on establishing an Institute-wide timesharing initiative until 1963, when Project MAC was organized under the direction of Robert Fano. The project was funded by the Advanced Research Projects Agency of the Department of Defense (ARPA). Created in 1958, ARPA was

> directly attributed to the launching of Sputnik and to the U.S. realization that the Soviet Union had developed the capacity to rapidly exploit military technology. Additionally, the political and defense communities recognized the need for a high-level Department of Defense organization to formulate and execute research and development projects that would expand the frontiers of technology beyond the immediate and specific requirements of the Military Services and their laboratories.[14]

ARPA ushered in an era of abundant funding for university projects, offering far more in terms of funding than any other research funds at the time. Where institutions such as the National Science Foundation and the Three Services Program provided funding to research programs at the level of tens of thousands of dollars, ARPA was willing to throw millions into the creation and support of promising research efforts.

Figure 6: Robert Fano

J.C.R Licklider was then the director of the Information Processing Techniques branch of ARPA. From his days at BBN, Licklider had become extremely interested in timesharing and saw in it the future of computing. Through his position in ARPA, he promoted and supported work in timesharing at a number of laboratories and institutions across the country, quickly acquiring recognition as the "Johnny Appleseed of timesharing."[15] MIT was still struggling to find money to fund the proposed computing facilities suggested by the Long Range Computer Study Group's 1961 report when Licklider suggested to Fano that he start a big laboratory dedicated to the advancement of the field of computer science. Fano's 1963 proposal for the formation of Project MAC illustrated Licklider's influence on the computing community. "He [Licklider] was convinced, [that] on the basis of his professional experience as a psychologist and computer user, that on-line interaction with computers … would lead to major steps forward in the utilization of computers in both civilian and military applications…. And Licklider's contagious enthusiasm provided the essential catalyst."[16] The proposal was accepted and a contract with initial funding of $2,200,000 was awarded through the Office of Naval Research on behalf of ARPA.

4.1 A SOCIAL CONSTRUCTION

From the very beginning, Project MAC was a tremendous effort in social construction. Fano's vision was to bring together all of the groups from MIT that shared an interest in computers to

[14] http://www.darpa.mil
[15] Fernando Corbato. Interview conducted November 26, 2001.
[16] Jack Belzer et al., eds. "Project MAC", in Encyclopedia of Computer Science and Technology. p. 343.

form a single research community. Though his eventual goal was to start a laboratory that would be the computer science complement to RLE, Fano wisely understood that the faculty and graduate students he wished to recruit "would not have come if that had meant for them changing allegiances to their laboratories."[17] The brainchild of Licklider and Fano was then started as a project in order to facilitate bringing research staff from other MIT departments. As a project, MAC could be staffed by people from various departments and allow them to maintain their existing laboratory affiliations. Among other groups from around the Institute, the AI Group under Minsky was brought under the umbrella of Project MAC and specifically allotted a full third of the project's budget.

The actual acronym, MAC, stood for two things: "Machine-Aided Cognition, expressing the broad objective of the AI research program; and Multiple-Access Computer, describing it as a major systems development endeavor."[18] The dual meanings of the name indicated that two of the pillars of the project would be the development of timesharing systems and research into AI.

As director, Fano hoped to bring legitimacy to the field of computer science. "You have no idea the extent to which people believed Computer Science was not here to stay," said Fano, "I was trying to persuade people… I remember Jerry Wiesner [MIT President 1971-1980] who was a good friend of mine asking me what is computer science and as I was explaining he fell asleep on me, literally." [19] Apart from Fano's vision of the project as an educational tool, Project MAC also had four major objectives:

1) To foster on-line use of computers in a variety of disciplines, by providing facilities, technical support and financial aid to interested research groups.
2) To design and implement a new timesharing system, on the basis of experience gained from operation of the CTSS developed by the MIT Computation Center.
3) To support basic research in the computer sciences on the part of interested faculty and students.
4) To provide the research and development environment necessary to educate future generations of students so that they will be able to meet the manpower needs of the country in the computer field.[20]

4.2 Logistics

Project MAC was initially located on the 8th and 9th floors of Tech Square, a building now called NE-43. The ninth floor was reserved for the project's computer systems, which in 1964, included a modified IBM 7094, and a PDP-6 which was purchased by Marvin Minsky to support the work of the AI Group. The Project MAC 7094 ran the same version of CTSS that was running at the Computation Center, the only difference was that the machine was run in timeshared mode constantly, whereas the Computation Center machine was still run in batch-processing mode part of the time. Since the computer in the Computation Center was still being shared among a number of universities and institutions along the East coast and was therefore unavailable most

[17] Robert Fano. Interview conducted November 29, 2001.
[18] Belzer, p.339.
[19] Robert Fano. Interview conducted November 29, 2001.
[20] MIT. Laboratory for Computer Science 1961-1988. AC 268. Institute Archives and Special Collections, MIT Libraries, Cambridge, Massachusetts.

of the time, Project MAC's dedicated machine greatly increased the availability of computing resources to the MIT research community.

Project MAC was introduced to the world during a 6-week summer study in July and August of 1963. Summer studies were quite common at MIT and served the purpose of gathering the best experts in a particular field of interest for a period of intense brainstorming and collaboration. The Project MAC Summer Study attracted some 57 people from universities, industry, and government, with many of them in residence for the full 6-week period. The participants were given access to the terminals of the CTSS system at the Computation Center. The summer study succeeded in fostering valuable technical interaction and established working-level contact between many university, government, and industrial laboratories. In a sense, the summer study served as the launching pad for the overall ARPA program by establishing a community of people with a common interest in timesharing and on-line interactive computation.[21]

5. The MAC System

From the beginning of its existence, the second of Project MAC's four primary objectives was to foster on-line use of computer resources by providing a timeshared system, called the MAC System. In the early years of the Project, CTSS served as this system. During this time, another timesharing system, by the name of Multics (Multiplexed Information and Computing Service), was being developed to fulfill the third objective of the Project. Later, after development was completed and Multics was deployed, it became the primary timeshared system operated by Project MAC, and thus adopted the name of the MAC System.

5.1 The Compatible Timesharing System (CTSS)

Corbato, the primary designer and implementer of CTSS, had always believed that the computational resources of the IBM 709 should be available for everyone who wished to use them. However, the mediated interaction between the programmer and the machine and the logistical difficulties imposed by the batch processing system hampered the use of computing machines by the public. Corbato saw in timesharing a means to bring computing to the masses. The original implementation of CTSS was a "quick and dirty timesharing system…just to see if it could be done. Initially CTSS was more of a hack, but later we refined it."[22] CTSS worked in conjunction with the IBM 709 operating system. Its beauty lay in the way it could "steal" time away from the operating system to allow multiple users' programs to run simultaneously on the computer.

Figure 7: Fernando Corbato

In order to allow CTSS to run in conjunction with the original batch operating system of the 709, the design was limited in the breadth and types of changes that could be made while still being able to run in batch mode. Since CTSS was to share memory with the operating system, the size of the system was a key design constraint. The first version of

[21] Belzer, p. 344.
[22] Fernando Corbato. Interview conducted November 26, 2001.

CTSS carved out 5Kb of memory from the 32Kb available in the 709. The key feature of the timesharing-enabled system though, was a hardware modification that made interrupts possible. As a custom addition to enable timesharing, IBM installed an interval timer on the 709 machine which could interrupt user programs.

CTSS was first demonstrated in November 1961 at the Computation Center. Immediately afterwards, Corbato began preparing for a second release. This was motivated by the fact that in the spring of 1962, the Computation Center got the latest IBM machine – a 7090. The machine contained two 32Kb core blocks of memory, which was double the standard amount of memory on any computer at the time. Additionally, the greatest achievement of the 7090 was that it marked the transition between tape memory and disk memory. The initial CTSS used a tape memory for all user storage. Within one year though, IBM introduced the 7094, which, while similar to the 7090, contained a few more features. Its data channels could access memory and run simple channel programs to do I/O once started by the CPU. Furthermore, the channels could cause a CPU interrupt when the I/O finished. In addition, several instructions and a memory boundary register were added to the processor.

The modifications to the 7094 meant that the computer that CTSS ran on was now able to operate in two modes. The first core memory block held the CTSS supervisor, while the second core was used for user programs. The memory boundary register was used to prevent a user’s program from accessing the first core block. The two modes enabled a system named the “Fortran Monitor System” (FMS) to be run as a background user program in the second core. The FMS had controlled both the 7090 and then the 7094, when they were operated in batch mode. The ability of the CTSS to run FMS in the second memory space definitively proved that Corbato had achieve a compatibility between batch processing and timesharing and that the transitions between them were seamless.

5.1.1 Implementation Details

CTSS handled interrupts using an interval timer clock that the supervisor set before switching to a user. If the clock expired before the user program exited or called for input from the typewriter that was not available, then the interrupt would take the CPU back to the supervisor. The supervisor would then enter the scheduler, which would choose another user to run. There was also a signal called “quit” that a user could send from his typewriter, which told the supervisor to halt the execution of his program. The intention of this command was for debugging; the user could examine the state of the computation in the middle of the execution of his program.

Upon logging in, each user of CTSS received a virtual machine, which was controlled by the typewriter terminal and allocated CPU time through the dispatching and scheduling process described below. The processor had a supervisor/user mode bit; users always ran in user mode and were unable to do I/O directly. Instead, they had to call the supervisor. In the first release of CTSS, each virtual machine had a memory space of a size specified by the user, which was only limited by the memory size of the underlying machine. There was only one job in memory at any one time. When a user was about to run, CTSS would write out the entire memory of the previous user to tape and then read in the memory of the new user. This process was called “swapping”. The swapping algorithm of the second release of the timesharing system was a bit more clever. The system was careful to write out no more of the previous user's memory than necessary to fit in the next user. That way, a small interactive user that was scheduled in the middle of a long-running job would cause only a small bit of memory to be swapped out. When that user was finished only this small bit of memory had to be swapped back in for the long-running job to continue. This scheme became known as the "onion-skin" algorithm.[23]

CTSS had a multi-level scheduler that was intended to give short-running jobs priority over long-running ones in an environment where it was not known in advance which job would be short and which would be long. The motivation for this prioritization algorithm was to reduce memory swapping, which was a very costly operation. There were two concepts involved in dynamically determining which process to prioritize: dispatching and scheduling. The dispatcher provided a priority system for the processes in queue. The dispatcher always took a job from the first queue if there was one there. Otherwise, it would move on to the second queue. Thus, a job in the last queue does not get any attention until all the higher queues are empty. The scheduler always puts a new job at the end of the first queue. When a job on the first queue runs, it gets a timer setting of one “quantum,” which was approximately equal to one second.[24] If the timer expires before the job finishes, the scheduler puts the job at the end of the second queue and invokes the dispatcher. When the first queue becomes empty, allowing a second-queue job to run, it gets a timer setting of two quanta. Each time a timer expires, the scheduler moves that job down one queue; each successive queue gets a timer setting that is twice as long as the previous queue. The effect of this queuing and dispatching implementation was that short-running jobs got immediate attention whereas long-running jobs lost priority as they continued to run. However, when longer-running jobs finally did get control, they ran for increasingly longer periods. Users were quick to discover that they could trick the system into keeping their job a high priority by hitting "quit" and typing "START," which would command the scheduler to place the job back at the end of the first queue. When the system was heavily loaded, this was one a way of pushing a long job through.[25]

Another feature of CTSS was that state was maintained for users in between their interactions with the system. In order to protect the state of each user and ensure that no rogue programs could accidentally modify another user’s files, the concept of a file system was developed. The first file system was very simple: each user had one directory and all files created by that user were named in that directory. Hard drive space was allocated by quota: each directory had a limit on the amount of space the files in it could occupy. Originally, all files were accessible only to the user that created them. Corbato soon discovered that one of the primary features that users of CTSS desired was the ability to share their files with other people using the system. The second-generation of CTSS included a “PERMIT” command that allowed a user to specify a file, the name of another user, and “READ” or “WRITE”. Once permission was given, if another user wanted to make use of the file, he would first issue the “LINK” command, specifying the name of the owner and of the file. From then on, programs under the control of the second user could read or write that file, as if it were located in the second user's own directory.

[23] “Abridged CTSS Programmers’ Guide” in Technical Information Program TIP-TM; 113. Cambridge, MIT, 1968. p. 45.
[24] Jerome Saltzer. Email Interview conducted November 19, 2001.
[25] Ibid.

5.2 MULTIPLEXED INFORMATION AND COMPUTING SERVICE (MULTICS)

CTSS provided evidence that timesharing was possible and that there were practical benefits to having a timeshared system. Specifically, the system was used to promote further research on computer systems and facilitate efforts in Artificial Intelligence. CTSS was a way to make the computer a public utility, but at its peak it could only support approximately 30 simultaneous users. Corbato envisioned a much larger system that could support hundreds of simultaneous, remote users. Later, he said that a computer utility was

> a computer, its software, and staff set up to provide 24-hour service for all of a community's information needs, Multics was envisioned as quite this, a “Multiplexed Information and Computing Service”, …, to be present, reliable, powerful, and all that is needed as an information resource for a large number of people, all the time. This was radical at the time, when computers were viewed mainly as tools and toys for scientists.[26]

Multics, a joint project of MIT’s Project MAC, Bell Telephone Laboratories, and General Electric, was designed to bring timesharing to the masses “the Right Way”.[27]

5.2.1 IMPLEMENTATION DETAILS

The Multics system achieved a number of important technical advancements, many of which were so groundbreaking that they became a part of standard operating system design in the years to follow. Among others, these innovations included an advanced memory architecture, high-level programming language implementation, robust security system, and dynamic reconfiguration.

The Multics memory architecture divided memory into segments. Each segment had addresses from 0 to 256K words (1 MB). The file system interfaced with the memory access system such that programs accessed files by making memory references. A very large breakthrough in the Multics development was the introduction of virtual memory, which allowed each user to believe that he had full control of the entire memory space.[28] The CPU generated addresses, which were then translated by the

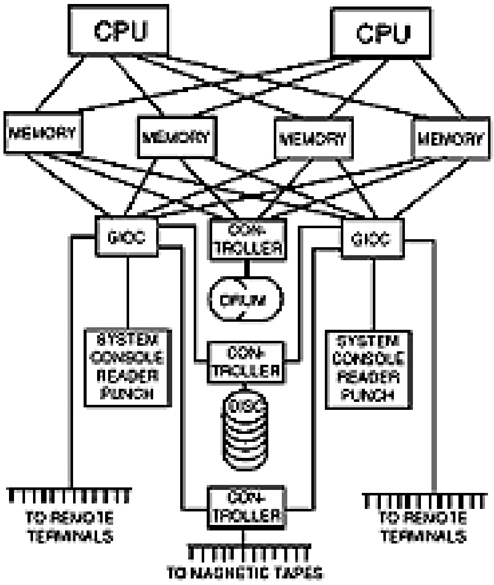


Figure 8: Multics hardware block diagram

[26] http://www.multicians.org/mgc.html
[27] Jerome Saltzer. Interview conducted November 29, 2001
[28] F.J. Corbato, and V.A.Vyssotsky. “Introduction and Overview of the Multics System”. Proceedings of the Fall Joint Computer Conference, Las Vegas, Nevada, November 30, 1965. at http://www.multicians.org/fjcc1.html

hardware from a virtual address to a real address. A hierarchical three-level caching scheme was developed, which used main storage, a paging device, and a disk to provide transparent access to the virtual memory.

Another achievement of Multics was that it was written in a high-level programming language instead of in Assembly, the language of CTSS. PL/I, developed by IBM, was chosen as the programming language in 1964. Permission from IBM had to be obtained to reprint their language manual. It soon became clear that the full PL/I language was harder to implement than had been expected. Corbato agreed to outsource the development of a compiler to Digitek. The language was so complex that after a year the contractor had not produced a compiler. Two Multics programmers, Bob Morris and Doug McIlroy, created a backup plan for a PL/I compiler. This language was called EPL (Early PL/I)[29]; the compiler produced output in EPLBSA (EPL Bootstrap Assembler). Even though the compiler was very slow and language features were limited, it allowed the programmers the chance to write much more sophisticated programs greatly enhancing the feature set of the system. Eventually, the compiler improved enough to be a real asset to the developers because it gave them the tools to remain focused on the implementation of the conceptual ideas instead of focusing on the optimization of code.

Security was one of the basic design requirements for Multics. The experiences of CTSS had afforded the designers the opportunity to hack together a security model. With Multics on the other hand, Corbato was determined to implement security right. At that time in the mid-1960s, all then-existing computer systems could be cracked; that is, their file access controls could be defeated and any user who could run a program on the machine was able to take over full control. Corbato wanted to build a system whose access controls could not be bypassed. Thus, features such as access control, supervisor integrity, and passwords were included in the initial design. Access Control meant including all of the steps needed to determine whether a user had permission to do that which he was requesting, be it read a file or run a program in a particular memory space. For example, if a user wanted access to a particular file, the user identity had to first be ascertained and then permissions on the file had to be verified. Users established identity at login by verifying a username/password combination with a master record. Interestingly, many of the early security breaches of Multics were various forms of unauthorized access. Once identity was established, it was checked against the Access Control List (ACL) of the item being accessed. An ACL was a list of users and group names in combination that specified individual and group access rights to a file. Such security features later helped to sell Multics to customers concerned about security, such as military and government sites.

As part of the computer utility ideology, Multics was designed to be able to run 7 days a week, 24 hours a day. So, CPUs, memory, I/O controllers, and disk drives had to be able to be added and removed from the system configuration while the system was up and running. These constraints led to the concept of Dynamic Reconfiguration, which today is called “hot swapping”. Dynamic reconfiguration meant that the same physical machine could be used in both a production and development environment simultaneously. This was very important to the designers of Multics because they did not have enough money to buy separate production and development machines. Dynamic reconfiguration allowed the Multics team to provide computing services at the same time that the system was being developed. In addition to this

[29] http://www.multicians.org/mgc.html

hardware hot swapping, diagnostics could be run on a CPU before adding it back into the system. I/O reconfiguration was later added as well.

Initially the debugging of the system code was a core dump from memory. Booting early versions of Multics took upwards of an hour, and since printing a core dump to a line printer took another one to two hours, development progressed somewhat slowly in the first seven months. Paul Green, an MIT developer, remembers those early days of debugging:

> The first day or so that we had FDUMP [program to dump core memory to disk] was the day we set a record for the number of crashes in a single day! (My memory says 13, but I have no way of validating that; it was certainly more than 6). Printing the dump to paper took 60-90 minutes, as I recall. This limited the number of crashes one could take per day. FDUMP sped things up so fast that we could crash more often. People didn't consider it progress at first; not until the software was made more reliable. Looking back on it, it is amazing that we tolerated the poor reliability and availability figures of that day. Our test procedures were really poor.[30]

More sophisticated debugging processes, including multiple debuggers for the different modules, were soon implemented for Multics. Object segments, line printers, and language debuggers[31] all existed to provide specific feedback to the programmer on the nature of the problem. Sometimes, these debuggers did not provide consistent information and it was hard to figure out the nature of the problem. Though a more comprehensive debugger was eventually built to satisfy the problem of debugging the system, this solution was not available for many months after development had started.

5.3 Systems Group Culture

With the second release of CTSS and the beginning of the development of Multics a rigorous structure began to cement around the development of the systems. A distinction emerged between those who designed the system and those who implemented the system modules. Corbato led a team of three people who were responsible for making all designs concerning system design. These three people also did the majority of the programming on the second release of CTSS. Corporate collaboration meant that the development of Multics to fulfill the ideal of a computer utility was fundamentally different from that of CTSS. However, since people who had designed and implemented CTSS were also key to the development of Multics, there are strong parallels between the culture of the CTSS development team and that of the Multics team.

Multics employed the same administrative hierarchy that had proven effective in the development of CTSS. The differences in the management styles of the two systems are only due to the large amount of money and resources invested in the development of the second system. Accountability, both in the administrative and design processes, was increasingly important as the task of managing three separate entities (MIT Project MAC, BTL and GE) added another layer of complexity to the management of the project. All administrative and policy decisions

[30] http://www.multicians.org/pg.html
[31] http://www.multicians.org/object-segment.html

were made by the “Trinity,” which included Bob Fano, Edward E. David of Bell Labs, and C. Walker Dix of GE. The “Triumvirate” was in charge of the design and implementation. This group was composed of Corbato, Peter G. Neumann of Bell Labs, and A. L. Dean of GE. Jerome H. Saltzer, an MIT graduate student, and Edward L. Glaser, also from MIT, were consultants to the Triumvirate.

The staff of Multics was composed mainly of paid profession programmers, employed by one of the three institutions involved in the project. Though the staff for the project increased dramatically over its lifetime, very few graduate students ever got the opportunity to work on Multics because it was hard to find interesting research in a system that was moving at such a rapid pace.

Corbato believed that the computer was a public utility, like electricity. He was trying to provide a robust and reliable system for hundreds of disparate computer users. This environment meant that the operating system always had to be in control. The programmers were expected to follow some rules – they could not just do whatever they wanted. To that end, anyone who wanted to add a feature to the system had to present it formally to the Triumvirate. While process would seem to discourage modifications to the system, Corbato argues that they “were a pretty liberal jury”.[32] Still, the development team of Multics put much effort into finding ways to build the system in a disciplined way. In many ways, the development of Multics provided the model that is still used today in the development of production-level software systems. High level languages, design and code review, structured programming, modularization and layering were all extensively employed to manage the complexity of the system, which was one of the largest software development efforts of its day. The *Multics System Programmer's Manual* (MSPM) was written before implementation started. The manual totaled about 3000 pages and filled about 4 feet of shelf space in loose-leaf binders.

The development of Multics was signature of the Project MAC enterprise. It was a solid piece of production quality software that had radically changed the model of computing. Tom Van Vleck ultimately remembers Multics as having been built in its own way.

> We used to speak of building things “the Multics way” To me this meant trying to solve the general problem rather than the immediate one; open discussion of every technical point; continuous evolution of our process; and attention to consistency and elegance. Doing it right rather than doing just enough. When the Multics hardware is obsolete and the code lost, I'll still prefer to develop software the Multics way.[33]

6. AI GROUP

Minsky, with visions of his own AI Laboratory at MIT, had immediately seen the advantages of a marriage with Project MAC. The funding the AI Group received as a part of MAC dwarfed by two orders of magnitude the funding it was receiving from its original supporters. Though ARPA typically left spending decisions up to the Directors of the projects it was funding, in the case of

[32] Fernando Corbato. Interview conducted November 26, 2001.

[33] http://www.multician.org/tvv-bio.html

Project MAC, the AI Group was guaranteed a full third of the yearly financial award. Though much better funded, the AI Group of Project MAC was simply the AI Group of the Computation Center and RLE transplanted to Tech Square. The group was rather small and took up only half of the eighth and ninth floors, which they shared with the Project MAC 709 computer and a good portion of the Project MAC computer staff. According to Tom Knight, "there was a clear division even at the time that Project MAC was founded between the AI weirdos and the rest of Project MAC, and that persisted."[34]

6.1 GROUP COMPOSITION

The AI Group attracted people of two backgrounds: graduate AI researchers who were genuinely interested in furthering the field of computer intelligence; and hackers from the Tech Model Railroad Club. The former group was the primary research body. These students worked on projects such as symbolic representations of knowledge, robotic vision and movement, and natural language constructions. For the most part, the graduate AI researchers were located on the eight floor of Tech Square, isolated from the machines and the hackers on the ninth floor. The hackers built the infrastructure and computer systems, which enabled the research group to press forward. Minsky outfit the lab with another PDP-1 and maintained the same open environment in which anybody could walk in the lab and play with the computer. Gerald Sussman, now an MIT professor in the Electrical Engineering and Computer Science Department, came to MIT as an undergraduate and, having played with computers in high school, ended up joining the Tech Model Railroad Club. An upperclassman told him about the computers that anybody could use over in Tech Square:

Figure 9: Joel Moses and Patrick Winston

> So I wandered over there [to Tech Square] and they didn't seem to mind me playing on the computer. So I wrote a program that learned to play tic-tac-toe using a neural net idea I had learned in this [Minsky and McCarthy's] book. I made it a random neural net and it varied the weights and... it sort of worked. One day this bald headed guy showed up, it was obviously Minsky but I didn't know that… it was obviously the guy in charge who was going to throw me out, and he walked up and said, 'Gee, what are you doing?', and I explained to him what I was doing. And he said, 'How would you like to get paid to do that kind of stuff?' So that was nice and I became the equivalent of a UROP, though that didn't exist yet and the second thing he said was, 'Why did you make the net random?' and I said, 'Well, I didn't want it to have preconceived misconceptions', and he said, 'Well it has them, its just that you don't know what they are.' And that was the smartest thing I had ever heard in my life and I became attached to this group of people by that phenomenon; it was completely accidental… That's the way it was

[34] Tom Knight. Interview conducted November 26, 2001.

> at the time: people who were interested in the stuff, showed up, played with the toys and if they were good they just sort of got staffed.[35]

Despite their lack of background and all-encompassing enthusiasm for computers, many of the hackers retained interest in the field of Artificial Intelligence. Many of them were heavily involved in building and experimenting, if not so much in the theory. For the most part, this cross-interest was unique to the hacker part of the group, the AI theorist and grad students "did not necessarily see the process of computing as a joyful end in itself, they were more concerned with getting degrees, winning professional recognition, and the ahem, advancement of computer science."[36] There was some amount of tension between the older researching staff and the hacker kids. David Silver, a 14-year-old when he first started hanging out with the AI Group, managed to annoy the researchers and grad students when his unscientific hackerism yielded results that were elusive to the scientific half of the group. Another of the hackers, Richard Greenblatt, a freshman, implemented a chess program that was even able to defeat some human opponents. The hackers had their own opinions of the research staff, believing that the graduate students were ignoramuses who just sat around on the eighth floor blindly theorizing about what the machine was like. Tom Knight, a hacker who joined the AI Group in 1965 as a freshman, describes the differences between the two parts of the group: "you can either worry about the recognition or worry about having fun and getting things done… there was a set of people who viewed computers as tools, not as an elegant jewel which needed to be polished and improved."[37]

6.2 The Hacker Culture

The hacker culture of the AI Group had a number of key characteristics: youth, obsession with computers, unique relation to fast, interactive machines, idea of the right thing, and sponsorship of free information. The hackers that were drawn into the group were very young; many were MIT undergraduates. The AI Group included a good deal of undergraduates at a time when most research projects were primarily staffed by graduate students and faculty. This group of kids was not only responsible for the systems development and engineering of the group, it was also in charge many of the important functions. Joel Moses, who joined the AI group as a graduate researcher, but later became interested in the computer systems component as well, pointed out that Minsky "had a lot of undergraduates… and he put the undergraduates in charge of everything and some of them dropped out and became full-time staff members."[38] Minsky's faith in his young staff is exemplified by Sussman who, as a freshman, was part of the committee that was responsible for selecting a

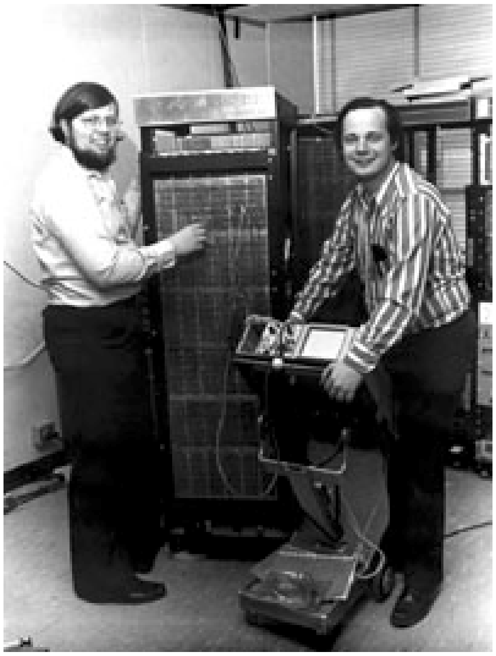


Figure 10: Tom Knight and Richard Greenblatt

[35] Gerald Sussman. Interview conducted November 15 and 29, 2001.
[36] Levy, p.72.
[37] Tom Knight. Interview conducted November 26, 2001.
[38] Joel Moses. Interview conducted November 29, 2001.

company to make a 1 Mb core memory for the group's system – a purchase which would eventually cost $400,000. Greenblatt too, quickly became the leader of sorts among the hackers. The hackers were not required to submit project proposals that involved broad system-wide changes; they did not even need to notify their superiors. The only channel that they had to go through was themselves. They even formed an organization called the Midnight Computer Wiring Society, which would, when it became necessary, circumvent the regulations of MIT against unauthorized tampering with expensive computers.

A second component of the hacker culture was their absolute passion for their work. In many instances, the hackers' obsession with computers was more important than progression in school. The most obsessive hackers "were so involved in the engineering of the thing that they didn't have time to even think about other things at all."[39] Greenblatt failed out of MIT because he spent all of his time at Tech Square, living thirty-hour days and sometimes ignoring his own personal hygiene for the sake of his mission to hack. There was even a joke around the AI Group that there was a new scientific measure for odor called the milliblatt. In order to decrease the measure of milliblatts in the lab, the group of hackers would need to maneuver Greenblatt to a place in the hallway of Building 20 where there was an emergency shower for cases of accidental chemical exposure.[40] Tom Knight also let his education at MIT slip, as he spent all his time on the eighth and ninth floors of Tech Square. Though he eventually did earn his degree, it took him about ten years. There were also others like Sussman, who managed to balance their hacking lifestyle with the needs of their educational program. Starting as a freshman, Sussman completed his full educational track and eventually earned his PhD with the AI Group.

Their unique perspective of the computer also characterized the hacker group. They were accustomed to fast, responsive, and interactive systems that were all their own when they were given time on the computers. Used to the operating style of RLE with the original PDP-1, "the fast-response systems made it possible to use the computer itself to help [the staff] compose text and programs, and this became the normal modus operandi of our staff and students."[41] Timesharing was then anathema to the entire hacker way of computing. The systems part of the AI group argued that man's time is more important than the machine, whereas the timesharing examples that were being implemented and used at the time, specifically CTSS, seemed to inconvenience the human operators for the sake of making the machine's job easier,[42] At best, timesharing was only "an apology for the fact that [they] couldn't all have [their] own machine."[43] The hackers felt an intimacy with computers that they thought was akin to the kind of relationship a person might have with his wife. "Timesharing was the equivalent of being with your wife, while sharing her with six other people simultaneously. And you don't share your wife with strangers!"[44] In a way, the hacker view of computing was decades before its time. The hackers envisioned a world in which they each had their own machine on which they would do whatever they wanted. Their ideas and values are largely echoed today in the personal computer, which at that time would have been so expensive that the very thought was preposterous.

[39] Gerald Sussman. Interview conducted November 15 and 29, 2001.
[40] Levy, p.58.
[41] Marvin Minsky. Email Interview conducted October 25, 2001.
[42] Tom Knight. Interview conducted November 26, 2001.
[43] Ibid.
[44] Levy, p.62.

In addition to their strong beliefs about how people should interact with computers, the hackers also believed in something called “the right way”. There was an aesthetic to programming that should not be violated and there were practices that were better than others. Kenneth Harrenstein’s thesis exemplified the attitude of the hackers, using language that indicated his belief in “the right way” to get things done. In discussing the computer system implemented by the AI hackers, he indoctrinated the belief that “anything which can be done by a user program should be kept out of the system code” and then introduced his implementation for a messaging application with the words, “as any good programmer knows…”.[45] Tom Knight’s analysis of the CTSS and Multics debugging practices also illustrates the hacker’s idea of “the right way:”

> The debugging technology on Multics and CTSS was pathetic. Core dumps basically was the technology. We never did core dumps… I was continuously amazed at the primitive techniques, which were being used to debug Multics. To this day I do not understand... Noel Morris would come in in the morning... and he would attempt to boot Multics. And booting Multics at that time was like an hour or something, I mean I don't know what the hell Multics did when it booted, but boy, it just took forever. So you'd boot Multics and then you'd type two, three commands at it and it would crash. And then, of course, the only next thing you could do is you could take a core dump. So for the next two hours you would dump core on the line printers. The major technical advance periodically was that you would get a faster line printer… I kept trying to tell Noel that there was a better way... And then what would happen is that Noel would take the dumps and spend the rest of the evening with, I swear, dumps that were four feet high of line printer paper. And he would look at the line printer paper... We would never have done anything like that, ever… There's two different things you can do when a system crashes: You can take a core dump and print it out and hope that you captured all of the information that was important and reboot the machine and get it going as fast as possible. That was kind of the CTSS and Multics theory - try to get the machines up as fast as possible, don't worry about the bugs... The thing that we did was that if the system crashed, there was usually someone here who was very competent who would start up the debugger and figure out exactly what was going on... The philosophy of when it’s broke, you fix it.[46]

Whereas the CTSS and Multics development teams were forced to use the primitive method of core dumping to debug their systems, the hackers found a much more elegant solution in DDT, their interactive debugger. They also used a number of hardware safety mechanisms, which would prevent users from causing harm to the system in addition to alerting the hackers to possible bugs in the system software. The hackers were all brilliant programmers that not only shared a genuine obsession with computer systems, but also had a strong sense of how they should be implemented.

The final characteristic that the hackers of the AI Group shared was the belief in a free flow of information. The group believed that they should be given access to the tools and resources that

[45] Kenneth Harrenstein. BS Thesis, EE, 1976.

[46] Tom Knight. Interview conducted November 26, 2001.

would facilitate their mission to find out and improve the way the world works. Property rights were mere obstacles to be circumvented when one of the group needed something to help him build, understand, or fix. High-security locks and vaults were only mechanisms that provided a false sense of protection to the people who believed they were locking everything up and preventing theft and the flow of information from going in the wrong direction. The hackers carried pounds and pounds of keys that would gain them access to every conceivable place in the building. The youngest of the group were sometimes assigned the task of crawling through the ceilings of Tech Square late at night to break locks or set up access systems. The access rights they bestowed upon themselves were never used to steal or injure, but were simply a means to get at critical information and resources. These ideals of free access and flow of information are reflected in the computer system that the hackers eventually developed for the AI Group. The system boasted that it had no security. "Everybody knew that Stallman's password was carriage return."[47] and individuals were allowed to access and change any file on the system, regardless of the owner. The hackers used to enjoy breaking into CTSS and leave little reminders to the system administrators that they had gotten past the supposed security system. A common hack involved breaking in the system, printing out a list of the users and passwords for the system, and then shoving the printout under the door of an administrator for them to find in the morning.

6.3 AI GROUP MANAGEMENT

Marvin Minsky and Seymour Papert comprised the management team of the AI Group until 1970. Minsky remembers, "we worked so well together that, for a decade, we each could run the laboratory effortlessly, leaving the other to decide what must be done; co-directing is never having to discuss non-technical matters."[48] Though Minsky was the original visionary who saw the AI Group one day being its own laboratory, he was always more compelled to study science and do research than engage in the bureaucratic activities required to keep the group functioning. Papert was even less of an administrator. Together, Minsky and Papert had "neither the talent nor the interest in management"[49] and were only too happy to forfeit their positions as co-directors to someone else after the AI Group became its own laboratory. The administrative needs of the group were covered by the multitude of tasks assumed by people like Russell Noftsker, an engineer and manager. Noftsker was responsible for most duties that kept the group running on a daily basis.

Figure 11: Marvin Minsky

Though they were not interested in the red-tape that always comes along with the administration of a sizable and well-funded organization, Minsky and Papert managed to create and maintain an environment of innovation, thus enabling the group to make substantial contributions in systems development and AI research. Tom Knight recalls, "Marvin Minsky was remarkably effective at fighting the bureaucracy. If you wanted to get something done and it looks like there was even a plausible reason why it would be a good thing, or even why it wouldn't be a bad thing, then he

[47] Joel Moses. Interview conducted November 29, 2001.
[48] Marvin Minsky. Email Interview conducted October 25, 2001.
[49] Patrick Winston. Interview conducted November 6, 2001.

would probably figure out a way of making it happen."[50] As a supervisor, Minsky was fairly hands-off and allowed his graduate researchers to do what interested them. His supervision consisted only of occasional meetings with his students, in which they would tell him what they were doing and then he would blow them away with his thoughts on something else. The hackers were left to organize themselves and engage in projects as they saw the need and interest. Joel Moses, a graduate student under Minsky at the time, describes the role of the hackers:

> They built the systems. We needed a display system; Tom Knight built a display system. We needed an operating system for the PDP-6; Greenblatt sat down with Nelson and they wrote an operating system. We needed something that was going to control robots; and they built a real-time system to control robots, they helped build robots.[51]

The loose administration of the lab is evident in the lack of memos and documentation about the activities of the AI Group while it was a part of Project MAC. Despite the tremendous effort that was being put into building the systems to support the research, only one thesis, written in 1976, was produced that is in any way related to this work. All other theses coming out of the AI Group in those days specifically related to research in AI, performed by the graduate research staff that was only loosely affiliated with the hackers. Documentation about the computing infrastructure of the group was generated only for internal reference and never for publication. This paucity of records can be attributed to the hacker's point of view that such documentation was unnecessary and boring to produce. If they wanted to understand something, they would much prefer to open it up and get right down in it to opening a manual and reading about it.

> Also there were other series of "working papers" that were regarded as entirely internal, for use by colleagues in the laboratory, and not advanced enough to become "AI memos", which often contained program listings, more details of how things actually worked, circuit diagrams, or even charts of where machinery was hidden under section of laboratory floor. But often there were substantial discoveries not documented elsewhere.[52]

Minsky wanted to foster an open environment of creation and innovation for the hackers. Since rigorous documentation was not a part of the culture, many of the great achievements in system design and implementation remain hidden in the minds of the hackers and the systems themselves.

The supervisors of the AI Group were very sympathetic to the eccentricities of their hacker prodigies. Minsky understood that the hackers and their ideals were what kept the lab productive and he knew not to tamper with the crucial components of hackerism. He was then sometimes forced to shelter his staff from the other administrators of Project MAC. Stu Nelson, for example, was always at odds with the rules and had a very volatile temper when it came to defending his hacker ideology. When he was caught red-handed hacking on the phone system, Minsky convinced his good friend, Ed Fredkin, to hire this incredibly brilliant nineteen-year-old

[50] Tom Knight. Interview conducted November 26, 2001.
[51] Joel Moses. Interview conducted November 29, 2001.
[52] Marvin Minsky. Email Interview conducted October 25, 2001.

with a knack for making mischief.[53] Knight recalls another instance in which Minsky saved the hackers from the unforgiving and watchful management of Project MAC:

> Dick Mills, the obnoxious financial person and kind of administrative type of Project MAC, at one point wandered up to the AI area and saw some beer cans in the trash barrel and said, ‘Gee there's not supposed to be any drinking here in the lab, Institute policy forbids it.’ Minsky said, ‘Oh no, those beer cans, they're not a result of anybody drinking, those are antennas, you see. You can make great microwave antennas...’ [He said this] with a completely straight face.[54]

6.4 The Incompatible Timesharing System (ITS)

While CTSS and Multics were being developed on the eighth and ninth floors of Tech Square, the hackers of the AI Group looked on with disgust. The timesharing implementation of CTSS was particularly appalling. At the heart of the issue was the question “which do you think is more important, the computer or the person? And in CTSS it was very clear that people's time was being wasted in order to make it easier for the computer to do its job and we always had the other point of view.”[55] Another common subject was the bureaucratic nightmare involved in the creation of Multics. “Can you imagine a project which was a collaboration between Bell Telephone Laboratories, GE, and Project MAC?” asks Knight. “It’s remarkable to me that they even got it off the ground...”[56] Knight also believed that regardless of the system objectives and development effort, the implementations of CTSS and Multics did not live up to the ideas upon which they were based. In any case, the AI Group initially looked upon Multics as the future of their computing infrastructure, hoping that the new system would provide more flexibility than CTSS to incorporate the necessary computing to do robotics work. Many of the graduate students in the AI Group even used CTSS on a regular basis to do their research, not sharing in the hackers’ disdain. Joel Moses emphasized this sentiment, “I started off using CTSS. I had nothing against CTSS. I was OK with CTSS.”[57]

By 1967, it had become disturbingly clear to the administrators of the AI Group that the Multics development would be a long, drawn-out process. The chances of Project MAC developing a timesharing system that would meet the needs of the AI Group in the near future were very low. Minsky realized that the AI Group would need a timesharing system of its own to facilitate research. More and more, the existing systems could not support the machine vision systems and robotics that were being developed in the lab. However, Minsky would need the systems core of the AI Group to implement a timesharing system and the hackers were all violently opposed to timesharing. By that time Ed Fredkin had rejoined the group and was given the task of convincing the group’s hackers to implement a timesharing system. Fredkin had a unique relationship with Greenblatt and managed, after months of wearing him down with suggestions and discussions about timesharing, to persuade him to do it. Legend has it that Stu Nelson, upon hearing of Greenblatt’s change in mind, was so upset by the thought of converting the group’s

[53] Levy, p.98.
[54] Tom Knight. Interview conducted November 26, 2001.
[55] Ibid.
[56] Ibid.
[57] Joel Moses. Interview conducted November 29, 2001.

beloved PDP-6 into a timesharing system that he threatened to destroy the computer.[58] Fredkin only managed to pacify him by guaranteeing that the system would be run in single-user mode during the late night hours. Corbato saw the AI Group’s timesharing system as “a self-serving response. In a sense, they wanted a system of their own. They didn't have to wait forever to see something come out the other end, and they deliberately built, in effect, a clubhouse system.”[59]

6.4.1 Implementation Details

Greenblatt and Nelson, in weeks of intense programming, produced the core of the system. To ensure that the machine ran as quickly as possible, and to minimize the performance loss in going to a timeshared system, the system code was written in machine language. As soon as some of the software was done, Tom Knight and a few others made the necessary adjustments to the PDP-6 hardware and added 1 Mb of memory, an unheard of amount at the time, to the existing system. The memory was nicknamed Moby because it was housed in a large cabinet with the girth of two Laundromat-size washing machines. Greenblatt and the hackers had full authority over the development and implementation of the timesharing system, which Tom Knight named the Incompatible Timesharing System (ITS). The name was a play on the Compatible Timesharing System (CTSS), referring to how the system programmers had completely scrapped the original batch operating system of the PDP-6 and written their own from the ground up.

From the very beginning, ITS was a bootleg project. Tom Knight explains, “we knew from day one that this had to be full duplex, character-oriented I/O, a whole set of things that you would never have found in CTSS, user-level interrupts... The goal was to provide fast interactive service to a small number of users.”[60] The people who worked on ITS were graduates of the PDP-1 school of Jack Dennis from the days back in the Research Laboratory for Electronics.

As with any other timesharing system, several users were able to run programs on ITS simultaneously. Nonetheless, ITS included some very unique features that set it apart from other contemporary timesharing systems such as allowing the same user to run several processes at once. Among other features, the system included an ability to provide real-time support for terminals and robotics attachments, a unique user interrupt architecture, and an unprecedented implementation of the system’s security. ITS afforded its users with extensive use of the displays and included what was at the time a very advanced system of editing that made use of the full screen. Moreover, the system could never have been acceptable to its hacker creators without providing at least the illusion that the computer’s resources were completely dedicated to the user. ITS featured full-duplex I/O whereas many generations of CTSS and Multics provided only half duplex I/O, which meant that the machine could not handle user input at the same time that it was outputting system responses and vice versa. Knight explains the interaction differences between the timesharing systems as “being basically one of do you want to provide an excellent experience, the best that one can arrange, for individual users, or are you more interested in increasing the number of people who can use the machine.”[61] The real-time, interactive system performance was then as much a social issue for the hackers as it was a technical problem. With

[58] Levy, p.114.
[59] Fernando J. Corbato. Interview conducted November 26, 2001.
[60] Tom Knight. Interview conducted November 26, 2001.
[61] Ibid.

CTSS, the experience of the interaction with the computer was not as important as allowing as many users as possible to use the system. With ITS, the hackers refused to sacrifice the nature of their interactions with the machine, be it in single-user or timesharing mode.

A crucial feature of ITS that set it apart from other timesharing systems was the way that it handled user interrupts. PCLSRing, pronounced "PC Losering," is a mechanism used by ITS to enforce a type of modularity when a process must access the state of another process. PCLSRing guarantees that no process ever catches another process in the act of executing a system call. Essentially, all calls to the system appear as atomic operations to the user. If an interrupt arrives while a process is making a system call one of two things will happen. The system call is completed, the interrupt is serviced, and the process returns to the next command after the system call, or the state of the system call is saved, the interrupt is serviced, and the process returns to the system call command to pick up where it left off. PCLSRing is not only a powerful aid in debugging programs, as it prevents the debugger from falling out of user code and into system code, but it is also critical to the system modularity of ITS. PCLSRing prevents processes from wresting control away from another process in the middle of a system call, thereby allowing processes to exchange control of the computer resources without corrupting their states. PCLSRing enabled users to run multiple processes from a single terminal with its support of terminal ownership in addition to making possible the highly interactive nature of the system.

A final design feature of ITS that is of particular interest is that of system security. Quite simply, ITS offered no security. The system did not use passwords and was designed to allow users access to any other user's files. The open architecture was designed to encourage users to look through other user's files to see what new projects they were working on, find bugs, and fix them. The computer programs would not belong to any individual, but rather to the world of users. In addition to the shared file environment, clever crossbar switching allowed any user on the system to find out who else was online and then switch himself to the terminal of a user he wanted to monitor or join in hacking. Interestingly, although every user of the system in the AI Group had the power to go in and destroy the files and notes of any other user, such malicious behavior did not take place. The system designers reasoned that if you made such things as file access and system crashes trivial, then no self-respecting programmer would ever be tempted to break in and wreak havoc on other users' files or on the system itself. Within the closed environment of the AI Group, ITS demonstrated that the best form of security was no security at all.

Despite its bootleg beginnings, ITS was sufficient for AI Group's needs during many years. The system was remarkably well designed and implemented, with the potential for evolutionary improvements built-in. Tom Knight points out that implementing the security measures used in most timesharing systems of the day would have been a relatively simple modification. These measures just never seemed to be necessary and were of no interest to the hacker implementers. Finally, ITS was exceedingly reliable. As Sussman remembers, "the PDP-6 computer on which this was built... was physically unreliable, mechanically and electrically unreliable... yet ITS was so well-written that it could run for months, fixing the errors as it went... It was built with reliability in mind."[62] Knight was also quick to add, "despite the fact that [ITS] was built on a house of cards hardware environment we ran the system for six to eight months at a time."[63]

7. The Widening Chasm

From the very beginning of Project MAC, the AI Group represented a significant portion of the project, receiving roughly one third of its total funding from ARPA. However, due to a number of reasons including differences in mission and culture, the AI Group and the rest of Project MAC never became bound to each other. By 1970, they depended on each other only minimally and they were not relevant enough to each other to necessitate being part of a single entity. The 1970 decision of the AI Group to split from Project MAC was due to a confluence of issues and only formalized a split that had already existed between the two groups for a considerable time.

7.1 Persistent Complaints

From the top level and down, the AI Group had always demonstrated significantly different attitudes about administration than did the rest of Project MAC. The philosophy of the AI Group's management, established by Minsky from the very start, was extremely flexible and involved little or no supervision. The graduate research students had formal meetings with Minsky only occasionally and the hackers were given complete autonomy to do whatever they saw was necessary with the group's computer systems. Though Minsky did manage to produce funding proposals and reports, he had no knowledge of the group's budget or daily administration. Addressing these administrative details was the job of people like Noftsker. The supervision of the AI Group was so loose, in fact, that Fano sometimes took on the job of scolding the hackers when they became too ambitious and overstepped their bounds. On the other hand, the rest of Project MAC was characterized by a much more structured approach to project management, as evoked in the development of CTSS and later Multics. Corbato and his team designated a small group of people as designers that had to approve and finalize all decisions before the designated programmers could begin any implementation. People such as Fano and Mills closely watched the administration of the various projects. These basic differences at the administrative level of the two groups undoubtedly produced tensions when the exploits of the freethinking hackers brought them into the territory of the other pet projects of Project MAC.

The presence of the hackers in the AI Group also resulted in a large cultural gap between those in the group and the rest of the Project MAC staff. Tom Knight remembers that this cultural division existed at the start of the project and persisted through the separation. The hackers of the AI Group shared an "idea of the right thing, there was an aesthetic which played a very important role … it was very clear in many cases that there was a right way of doing things and a wrong way, and we wanted to do the right thing."[64] Perhaps the root of this cultural difference stemmed from the fact that the AI hackers were mostly undergraduates who had naturally gravitated to the group because of their curiosity and shared enthusiasm for computers, whereas the rest of MAC was composed of faculty members, graduate students, and paid professionals more persuaded by career advancement than personal passions. While the AI Group also included a contingent of

[62] Gerald Sussman. Interview conducted November 15 and 29, 2001.
[63] Tom Knight. Interview conducted November 26, 2001.
[64] Ibid.

graduate researchers who were exclusively interested in AI as a computer science, the hackers' colorful characteristics, in many ways, defined the culture of the AI Group. These hackers often let progress in their formal educations slide in favor of spending more time with their computer obsessions. Winston characterizes this important difference in culture: "the AI Lab was hackers who rolled up their sleeves and just did it whereas the rest of [Project MAC] tended to be much more formal-type people."[65]

Besides managerial and cultural distinctions, the ideologies of each group were also in sharp contrast. As Winston observes: "the mission of Project MAC and later LCS was to advance the art of computation … Whereas the mission of the AI Lab back in those days was to develop an understanding of the nature of intelligence from a computational point of view."[66] The MAC Systems Group was interested in the application of an idea introduced years before but still required implementation in order to provide the luxury of computation to the masses. It was a tour-de-force to prove that a large number of users could successfully interact with a single machine. Minsky, however, was more interested in theoretical problems. He wanted to study the nature of intelligence, what made people capable of displaying intelligence, and how computers were useful in furthering this study. The AI graduate researchers were tackling such problems as computer vision, robotic movement, and symbolic interpretation of language. Even the hacker's were offering significant contributions to the AI research. Greenblatt's chess playing program was capable of defeating some of its human opponents. Winston described the objectives of AI research as "a big science mission, whereas advancing the art of computation was a big applied mission."[67] Even in 1970, after the Multics project had been farmed out of Project MAC, and the project was turning its focus to research to actual computational issues such as operating systems, programming languages, and security. Minsky's Turing Award lecture questions the legitimacy of such avenues of research:

> Almost all their time is devoted to formal classification of syntactic language types, defeatist unsolvability theories, folklore about systems programming, and generally trivial fragments of "optimization of logic design" – the latter often in situations where the art of heuristic programming has far outreached the special-case "theories" so grimly taught and tested–and invocations about programming style almost sure to be outmoded before the student graduates. Even the most seemingly abstract courses on recursive function theory and formal logic seem to ignore the few known useful results on proving facts about compilers or equivalence of programs… Until all this preoccupation with form is replaced by attention to the substantial issues in computation, a young student might we well-advised to avoid much of the computer science curricula, learn to program, acquire as much mathematics and other science as he can, and study the current literature in Artificial Intelligence, complexity, and optimization theories.[68]

[65] Patrick Winston. Interview conducted November 6, 2001.
[66] Ibid.
[67] Ibid.
[68] Marvin Minsky. Turing Lecture, April 1970. at
http://www.media.mit.edu/~minsky/papers/TuringLecture/TuringLecture.html

These differences between the science of the AI Group and the application of the rest of Project MAC are particularly exemplified by some of the interests of the CTSS group and the hackers who finally implemented ITS. Corbato intended to use CTSS as proof that timesharing was a viable alternative to batch processing to try and influence the way computer manufacturers thought about computers. Even though the first version of CTSS and Jack Dennis's PDP-1 timesharing system demonstrated the feasibility of timesharing in the early 1960's, the big players such as IBM continued to shy away from developing their own mainstream timesharing products. Thus, Corbato believed that CTSS was a tool with the potential to influence the computer industry. Clearly then, CTSS and later Multics were serious projects led by professional programmers on a very important and public mission. This larger interest led the CTSS and Multics teams to venture into the politics of the large computer manufacturers was something the AI group would not even consider as within the scope of their duties. Least of all the hackers who were more concerned with producing wildly revolutionary software for their own amusement than in changing the way IBM understood and marketed computers. Although they believed in the free flow of information, any organized effort to influence the way companies worked would have found no support from the hacker camp. The AI hackers detested bureaucracy and one of its main complaints had always been the high-browed and wasteful corporation, IBM. They did not want anything to do with it. ITS was developed with the goal of enabling the AI research staff to make full use of their robotic equipment and programs while maintaining the principles of computer interaction that the hackers so highly valued. Though the hackers believed it was the ideal timesharing system, they never intended it to become a utility for the common public. Saltzer best describes the difference in research approaches: "the system would support the research, the system wasn't the research."[69]

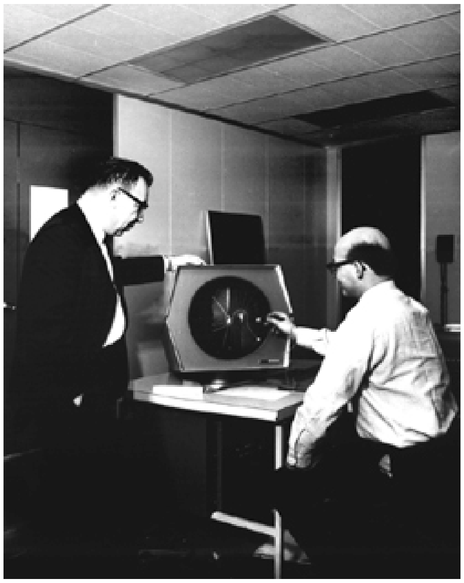


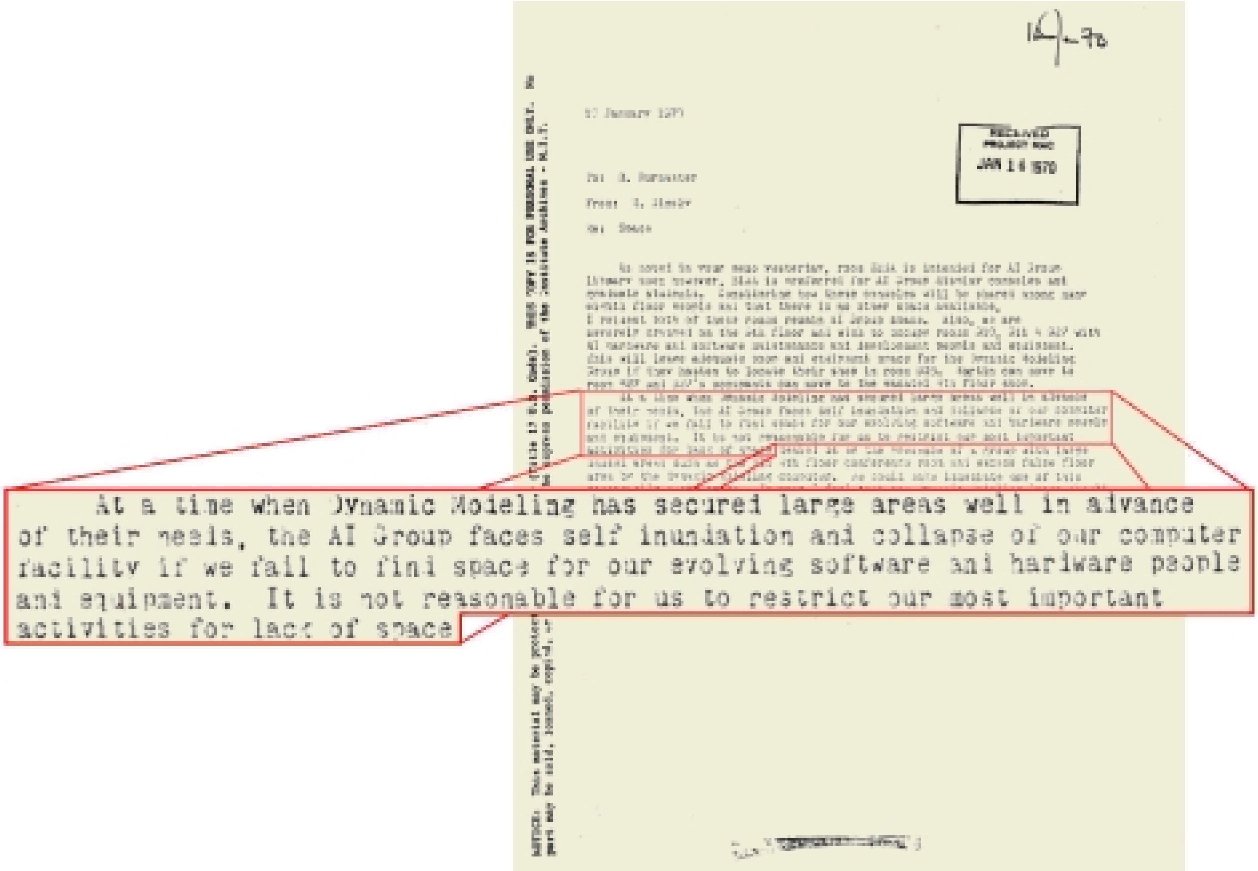
Figure 12: R. Fano and M Minsky playing Space Wars

7.2 INDEPENDENT INFRASTRUCTURES

The pre-existing differences in management style, culture, and mission laid the groundwork for a great divide between the AI Group and the rest of Project MAC that was further cemented by developments throughout the 1960's. By 1970, the supporting infrastructure of the AI Group, including the group's spatial location, funding, and computing systems was almost completely independent from that of Project MAC. Physically the AI Group was isolated from the rest of the project even before the official split between the groups. Both of them occupied different

sections of the building. Even on the 9th floor where the computers of both groups were installed, they worked in opposite corners of the building; interaction between the groups was not

[69] Jerome Saltzer. Interview conducted November 19, 2001.



14-Jan-70

13 January 1970

To: D. Burmaster

From: M. Minsky

Re: Space

[illegible]

At a time when Dynamic Modeling has secured large areas well in advance of their needs, the AI Group faces self inundation and collapse of our computer facility if we fail to find space for our evolving software and hardware people and equipment. It is not reasonable for us to restrict our most important activities for lack of space [illegible]

Figure 13: Memo from M. Minsky to D. Burmaster regarding space issues

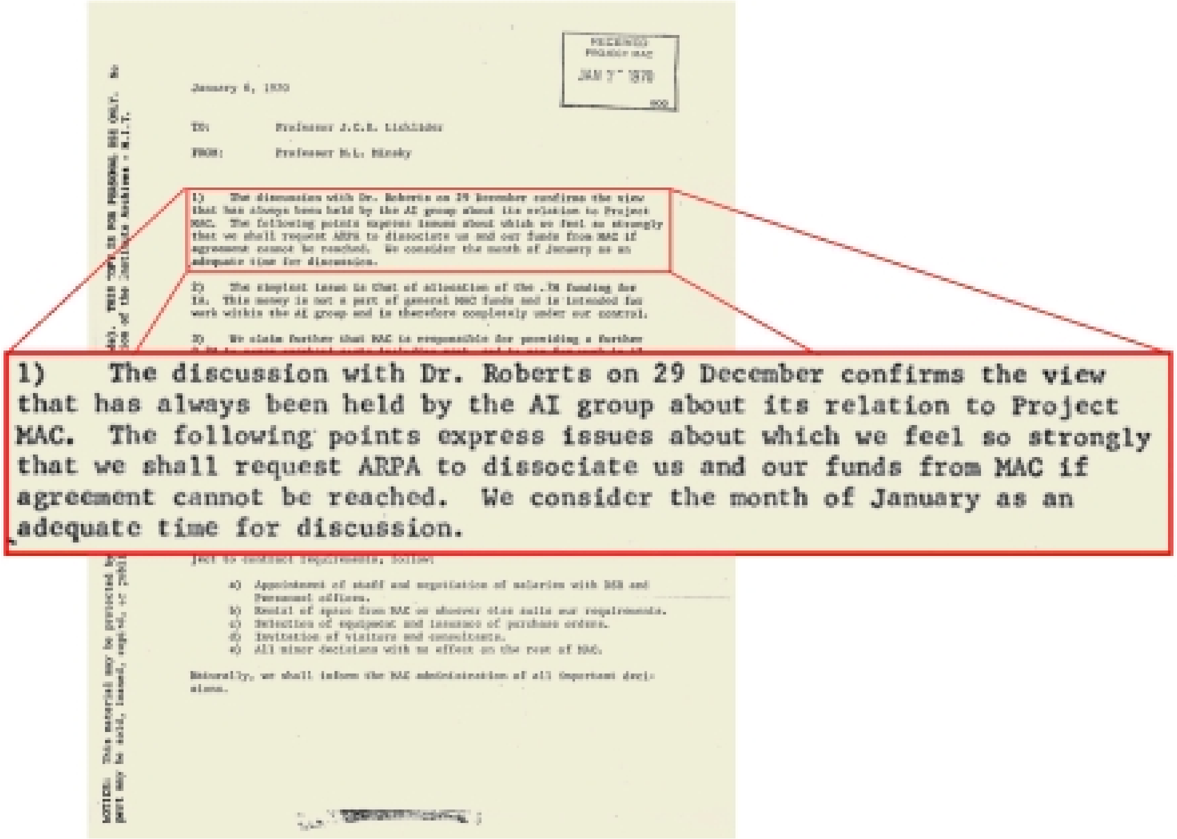
mandated by their relative locations. However, memos dating back to the time of the separation document Minsky's concerns that Project MAC's growth was crowding his group and taking away precious space that was desperately needed by the AI Group. For example, in this excerpt from a memo sent in 1970 from Minsky to David Burmaster, J.C.R. Licklider's assistant, he clearly expresses his stance on the situation:

> At a time when Dynamic Modeling has secured large areas well in advance of their needs, the AI Group faces self-inundation and collapse of our computer facility if we fail to find space for our evolving software and hardware people and equipment. It is not reasonable for us to restrict our most important activities for lack of space.[70]

Thus, in addition to being physically separated from the rest of the Project, the AI Group believed that their growth was being hampered by the expansion of other groups within Project MAC.

[70] MIT. Laboratory for Computer Science 1961-1988. AC 268. Institute Archives and Special Collections, MIT Libraries, Cambridge, Massachusetts.

Figure 13: Memo from M. Minsky to J.C.R. Licklider regarding funding



January 6, 1970

TO: Professor J.C.R. Licklider

FROM: Professor M.L. Minsky

1) The discussion with Dr. Roberts on 29 December confirms the view that has always been held by the AI group about its relation to Project MAC. The following points express issues about which we feel so strongly that we shall request ARPA to dissociate us and our funds from MAC if agreement cannot be reached. We consider the month of January as an adequate time for discussion.

1) The discussion with Dr. Roberts on 29 December confirms the view that has always been held by the AI group about its relation to Project MAC. The following points express issues about which we feel so strongly that we shall request ARPA to dissociate us and our funds from MAC if agreement cannot be reached. We consider the month of January as an adequate time for discussion.

2) The simplest issue is that of allocation of the .7M funding for 1A. This money is not a part of general MAC funds and is intended for work within the AI group and is therefore completely under our control.

3) We claim further that MAC is responsible for providing a further

ject to contract requirements, follow:

a) Appointment of staff and negotiation of salaries with DSR and Personnel offices.
b) Rental of space from MAC or whoever else suits our requirements.
c) Selection of equipment and issuance of purchase orders.
d) Invitation of visitors and consultants.
e) All minor decisions with no effect on the rest of MAC.

Naturally, we shall inform the MAC administration of all important decisions.

Figure 14: Memo from M. Minsky to J.C.R. Licklider regarding funding

The funding of the AI Group was also separate from that of the rest of Project MAC. Since the very beginning of the project, a full third of the funding provided by ARPA was designated for the AI Group, in addition to the funding that Minsky had already arranged for his group before joining the project. Saltzer remembers that the established precedent of separate funds was threatened in the late 60's by Multics: "Multics was soaking up all of the money... a huge amount of money was coming in and it was all being funneled into this, and the AI people were having trouble getting their hands on it. They saw this large amount of money going into this and couldn't see that it was important to them."[71] Following the decision to separate the AI Group from Project MAC, Minsky even addressed a memo to Licklider, then the Director of Project MAC, requesting that the funds for the AI Group be extricated from those of the rest of the Project MAC.

> The discussion… confirms the view that has always been held by the AI group about its relation to Project MAC. The following points [in relation to funding] express issues about which we feel so strongly that we shall request ARPA to dissociate us and our funds from MAC if agreement cannot be reached.[72]

[71] Jerome Saltzer. Interview conducted November 19, 2001.
[72] MIT. Laboratory for Computer Science 1961-1988. AC 268. Institute Archives and Special Collections, MIT Libraries, Cambridge, Massachusetts.

The memo describes how Minsky was unhappy with the lack of efficiency and speed in the Project's delivery of funds to the AI Group. Freeing themselves from Project MAC even offered the group the possibility of increased funding, since an independent laboratory devoted to research in AI would have a stronger presence than any group that was simply part of a larger entity that was known throughout the world primarily as a timesharing effort. The funding model used within ARPA at the time was that of giving large chunks of money to individual laboratories to be divided up at the discretion of the laboratory director. If the AI Group started its own laboratory, it could write proposals for itself and guarantee that the awarded money would go to research in AI. With the possibility of blanket funding devoted exclusively to AI research and the majority of Project MAC funds somehow being funneled through the Multics budget, the original advantages offered to the AI Group in joining the project were no longer evident.

With the creation of ITS, the AI Group's computing infrastructure achieved independence from Project MAC. ITS not only represented the hacker's solution to timesharing, it also gave the AI Group complete autonomy from Project MAC in terms of their computing environment. The group was no longer dependent on CTSS or Multics for their systems.

Clearly, the ties that bound the AI Group to Project MAC were weak. By the time it became its own laboratory in 1970, the AI Group was capable of providing its own supporting infrastructure and was no longer reaping enormous advantages from its marriage with Project MAC. This fact, combined with the differences in management, culture, and mission that pre-dated the project, made the AI Group a part of Project MAC only in name. The eventual separation and transformation of the AI Group into its own laboratory was merely a formality, and had little or no impact on the everyday work of the two groups. The actual event was so anticlimactic, in fact, that Winston, an AI graduate researcher, recalls, "Actually we grad students hardly even noticed… nothing really changed."[73]

8. Conclusion

The history of the emergence of the MIT Artificial Intelligence Laboratory, though interesting in itself, gains global significance as the story of a basic revolution in man-machine interactions. The concept of timesharing and its subsequent implementations reveals how people's relationship to the computer evolved into what it is today. Furthermore, as the nature of man-machine interactions evolved from the distant and limited interface of the batch processing machines to the real-time, interactive systems of ITS, the foundations for research into contemporary computer sciences were established.

8.1 Bringing It Together

The nature of man's relationship to computers has consisted of at least four major phases since the dawn of generalized computing machines. The first was the era of the mainframe computer, which was shared among organizations or people who had no other reason from being on the same computer other than saving money. These users had little or no interest in sharing their data with others on the system and for the most part had nothing to do with the functioning or

[73] Patrick Winston. Interview conducted November 6, 2001.

maintenance of the machine. Batch processing represents one example of the mainframe computing model. The second phase of computing models was the emergence of the mini-computer, which was shared only among small groups of people who are for the most part cooperative. The mini-computer model allowed users more control over the entire system and provided more intimate interactions with the machine. As computation power continued to become more accessible in terms of cost and size, the third phase of man-machine interactions, the personal computer, became possible. Under the personal computer model, the user owned and administered the entire system, including all of the files on the computer. Security was not needed in the personal computer model because only a single person used the machine and all of the files belonged to that one person. The final step in the evolution of man-machine interactions, the phase that is currently dominant, is the Internet, which allows the owners of primarily personal computers to provide any level of access they wish to others connected on the network.

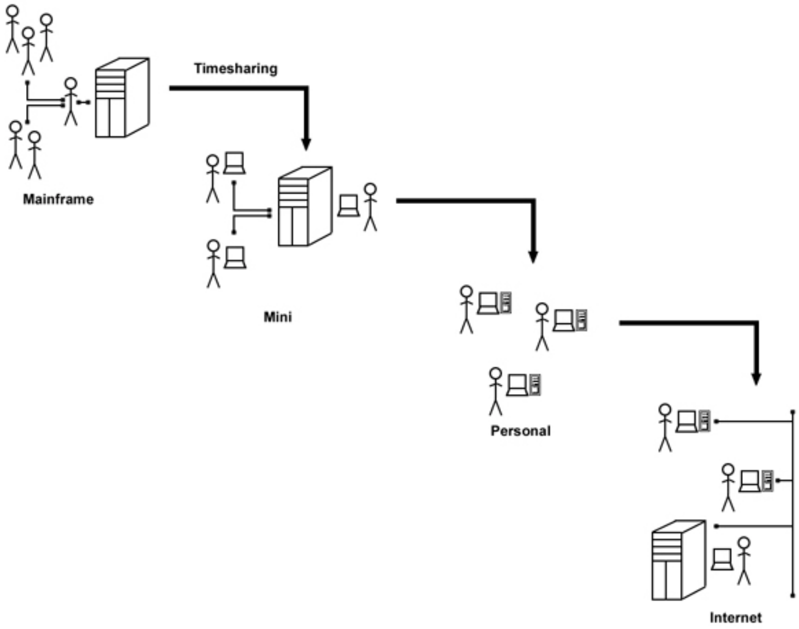
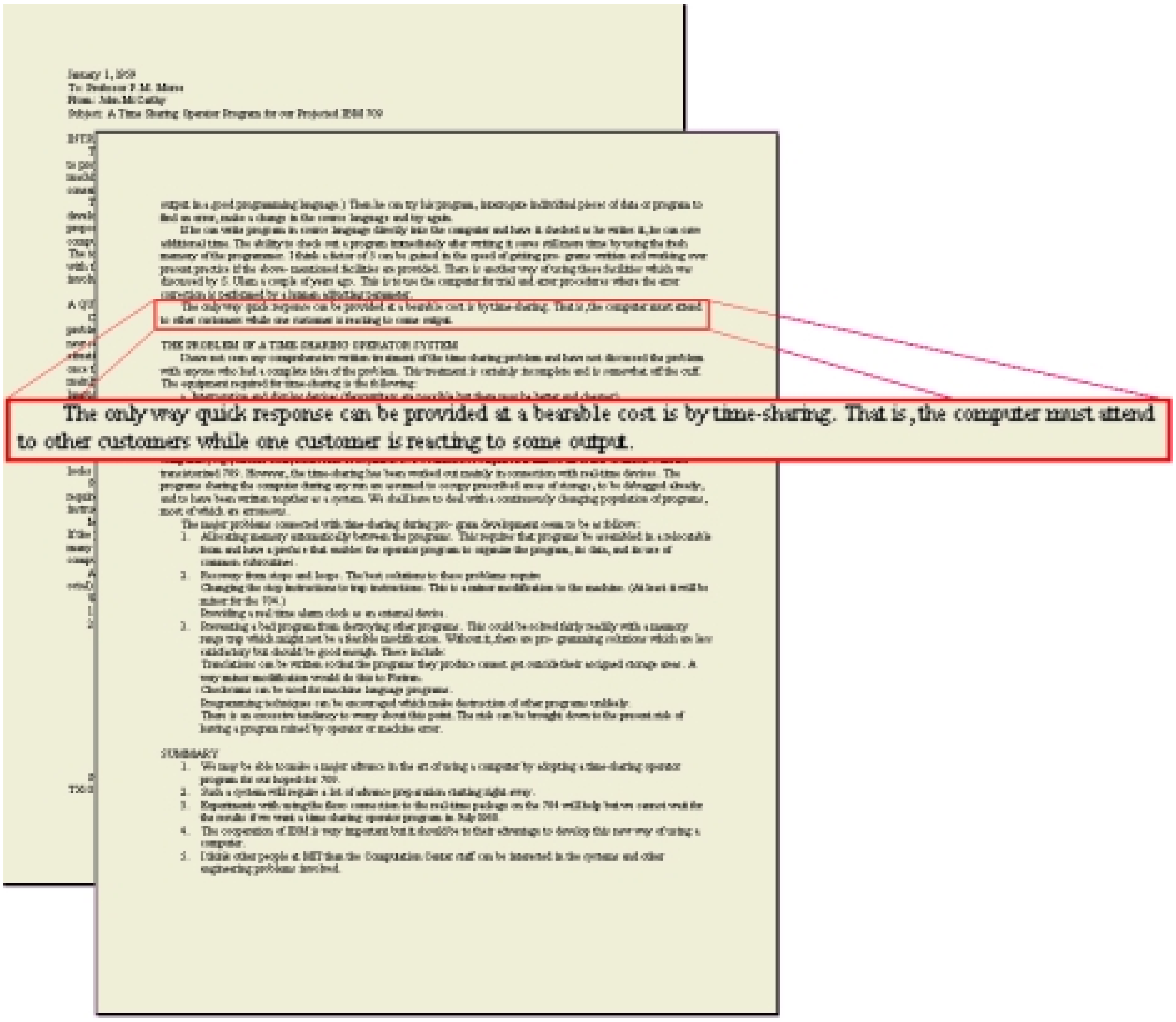


Figure 15: Evolution of man-machine interactions

In many ways, timesharing grew out of the frustration of interacting with computers at the batch processing level and was the vehicle by which the mainframe computing model gave way to the mini-computer model of computer interactions. As early as the mid-1950’s, the notion of timesharing was ambiguously understood as the idea of the computer somehow sharing its processing time among a number of different processes simultaneously. The modern conceptions of timesharing in which several users are provided interactive terminal access to a single machine simultaneously were not formally articulated until John McCarthy wrote his 1959 memo on timesharing to the Director of the MIT Computation Center, Philip Morse. McCarthy’s memo was a catalyst for a number of timesharing efforts at MIT in the early 1960’s. Even after McCarthy formalized the definition of timesharing in his memo, the variety and products of the MIT timesharing efforts demonstrate that the idea was still subject to a great deal of interpretation. Jack Dennis in RLE used the idea to allow people interactive access to the PDP-1 at the same time that it was monitoring and controlling lab equipment. Corbato, on the other hand, saw timesharing as the key to



January 1, 1959
To: Professor P.M. Morse
From: John McCarthy
Subject: A Time Sharing Operator Program for our Projected IBM 709

THE PROBLEM OF A TIME SHARING OPERATOR SYSTEM

The only way quick response can be provided at a bearable cost is by time-sharing. That is ,the computer must attend to other customers while one customer is reacting to some output.

SUMMARY

Figure 16: McCarthy's memo on timesharing

implementing his vision of the computer utility, in which computing power would be as ubiquitous and black-boxed to users as electricity. Later when Project MAC was established with a primary objective to provide increased access to computing to the MIT community, Corbato's interpretation of timesharing as a means to bring computing to the masses seemed to be the most obvious and applicable solution. Thus, Corbato drove the systems group in the Project down the path of developing the computer utility, eventually even spending over 5 years on the development of a production timesharing system that was the embodiment of his vision.

All the while that CTSS was adopted for immediate use by the MIT community and Multics was being designed and developed, the AI Group within Project MAC remained focused on their primary research interests in studying computer science. Specifically, the AI Group was interested in using computers to develop models of intelligence, such as symbolic languages and robotics, in the hopes that their research would elucidate the very nature of intelligence. The research of the AI Group necessitated more control of the computer than the model of the computer utility could afford them. The failure of CTSS and Multics to meet the computation requirements of the AI research, combined with the hackers' mandate that their timeshared computer system offer a commensurate level of interaction as a single-user machine, gave rise to ITS, which can be considered an implementation of the mini-computing model built on top of a mainframe system. From the very beginning, the goals of ITS were very different from those of Corbato's computer utility; ITS was intended to provide real-time interactive timeshared computing to a small community of users whereas CTSS and Multics sought to simply share access to a computing machine among a large number of unrelated users. The computer utility model thus improved upon the mainframe computing model of batch processing only along a single axis measuring accessibility. In contrast, the system designed by the AI Group, intended to facilitate AI research, followed a different trajectory of development in which the level of control and interaction afforded to the individual users of the system was at least as important a measure as the number of simultaneous users that the system could support. Finally, in contrast to the Project MAC Systems Group and their development of CTSS and Multics, the AI Group never viewed ITS as a research project; it was only meant to enable the group to do their real research on AI.

What is interesting about the story of the parallel development of timesharing systems in the Project MAC Systems Group and AI Group is that around 1970, as Multics was finally being given over to GE-Honeywell, the rest of Project MAC was far behind the AI Group in developing actual research into Computer Science. Whereas all of their efforts had previously focused on developing a product in an environment that was so fast-paced that there was little room for graduate research, they were at a loss for how the Project would perpetuate itself and fulfill Fano's vision of eventually becoming a research laboratory. Minsky's Turing lecture, anticipated this void created in the departure of Multics and named it as a reason to differentiate AI research from the rest of the MIT Computer Sciences of the day. These research avenues were mostly by-products of Multics system design, such as operating systems research, programming language development, and optimization theory. Around this time as well, the rest of Project MAC finally realized that to do actual research on computers, they would need to have more control of the machine than afforded by the system they had spent most of the past decade developing. Eventually, the rest of Project MAC joined the AI Group in their use of the mini-computer model of man-machine interaction. Professor Saltzer explains that while the computer utility model "is actually a very good environment to do other kinds of science that aren't involved in studying the computer itself," it is of little use in the actual computer sciences. "When you're trying to work on the computing system or the computer for computer science, now you need more control over how the computer works, and you'd like to be able to tinker with it or this and that and the other thing... And so you need your own,"[74] he argues. The rest of Project MAC eventually realized that which the hackers of the AI Group had known all along: a computer can not appear as a black box to the people who need to fix, study, and improve upon it. The computer utility model was thus of no use in facilitating the research into the computer sciences that the rest of Project MAC began after Multics development ended.

8.2 Observations On The Nature Of Engineering Revolutions

Perhaps what is most important about the emergence of the MIT Artificial Intelligence Laboratory and the development of timesharing are the insights this story provides into the nature of engineering revolutions. In addition to illustrating the nuances of black boxes as tools

[74] Jerome Saltzer. Interview conducted November 19, 2001.

in innovation and demonstrating how technological trajectories are pulled at and driven by a number of forces that are neither natural nor obvious, the sociological history also offers a number of observations.

First, it is clear from the parallel development of CTSS and ITS that past experience dictates future expectations. Both the hackers of the AI Group and the programmers of the CTSS, and later Multics, systems believed that they were implementing a timesharing system that was capable of providing multiple users interactive access to a single computer simultaneously. "Both the Systems Group and the AI Group were very strongly in the camp of doing it the right way," says Saltzer, adding "each was convinced that they knew the right way."[75] From a high level, both ITS and the MAC system achieved this common goal. Yet despite this shared objective, the differences between the interactions provided by the hacker system and those provided by the MAC system were so glaring that it is obvious that the implementers of CTSS and Multics had a different expectations for man-machine interactions than did the hackers. CTSS was grown out of the Computation Center IBM 704 batch processing culture and favored the utility model of computing. Users were independent from one another and small inconveniences in the man-machine interface such as half duplex I/O were still worlds better than no interface at all. On the other hand, the hacker culture, and the ITS system, reflected the computing culture that emerged from the Research Laboratory for Electronics where from the very early days, students were allowed time alone on the TX-0, which featured interactive debuggers that allowed for real-time program development. Timesharing to the hackers simply meant a clever way of multiplexing control of a single computer among several of the group such that the individual user experience was not at all diminished by the presence of other users on the system. Just as the experiences of the original CTSS programmers with batch processing clouded their judgment about an acceptable level of machine interaction, the AI hackers expected a more interactive relationship to computers because they had grown accustomed to single-user direct-access way of computing. The influence of the past also resonates in the research efforts of Project MAC after Multics development had ceased, which focused on studying further the issues and problems encountered during the development of the production timesharing system. Certainly, the experiences of Ken Thompson and Dennis Ritchie in the Project MAC Systems Group profoundly influenced their implementation of UNIX, which was essentially a mini-computer implementation of Multics. Similarly, the ideals of the hacker culture and the systems development practices of the AI Group echo clearly in the beginnings of GNU and the open source movement. Richard Stallman, a hacker who joined the AI Lab in 1971 and worked on ITS development, is often considered the father of the open source movement. His experiences within the hacker group clearly influenced his ideas about the value of information and the restrictions which should be placed on its consumption. Thus, technological developments and trajectories are in many ways products of the past experiences of the engineers responsible for designing and implementing the technologies.

Secondly, the founding of the MIT Artificial Intelligence Laboratory is a perfect demonstration of two very different approaches to engineering and development, which are both successful. The first tactic is the hierarchically organized model followed by the Project MAC systems team. This methodology was well in place in the systems groups even before GE and Bell Labs became involved in the development of Multics. Senior engineers and management had to approve of

[75] Ibid

design decisions before an army of lower-level engineers was allowed to implement them. This paradigm is the most favored approach to product development within modern-day corporations. Even at the lowest levels, the engineers are given few opportunities for innovation and out-of-the-box thinking. The alternative to the hierarchical approach is the unstructured and flat organization of the AI Group. This model is characterized by the isolation of the engineers and scientists from the bureaucracies and networks required for them to sustain their development environment. In the case of the AI Group, it is unclear who the heterogeneous engineer is behind the incubated environment of the lab. Minsky certainly cannot be classified as a heterogeneous engineer because his non-managing management style gave rise to a center of innovation and out-of-the-box thinking; such an interpretation would remove agency from the notion of heterogeneous engineering and thus render the classification a product of hindsight, rather than a product of the personality or actions of the engineer. The most likely answer is that there wasn't any single person that can be credited with creating the free environment of the AI Group. One possibility is that Minsky's genius was such that everybody around him took on a sort of enabling role, to ensure that he was given the resources and environment necessary to do his research. Undoubtedly, Minsky also had a hand in recruiting people for his group that he knew would provide their own organization. The hacker culture certainly has built into it a sort of self-sustaining quality. Linux and the persistence of the GNU project and open source are testimony to the survival skills embedded in the hacker culture. At the same time, people like Richard Noftsker managed the more administrative aspects of the AI Group. Thus, the story of the founding of the AI Lab offers some valuable insight into the two primary forms of project development.

Finally, this history is a testament to how closely related and intertwined are the social and technical aspects of engineering development. Just as Tom West was able to look into the DEC VAX and see the cumbersome corporate culture of the computer giant, the MAC System and ITS very much reflect the cultures and ideologies from which they were conceived. The structured and hierarchical development practices of the MAC System Group appear in the system as a well-planned modular design. Similarly, the industrial and corporate nature of the group developing Multics is reflected in the measures to which the system goes to guarantee security to the users. ITS, on the other hand, very strongly embodies the hacker ideals of an open and amorphous organization. Just as in the fluid social organization of the group, anyone could alter the basic computer system. Free-flowing information within the system design was guaranteed by the lack of any security measures. In many ways, the technical details of the MAC System and ITS were products of not only the technologies that were available at the time, but also of the social environment in which they were developed.

The boundary between social constraints and technological requirements becomes even more blurred when taking into account the goals and successes of the MAC System and ITS. Although ITS was of such enormous use in the computer science research of the AI Lab, it could have never served as a viable timesharing solution for the masses. On a social level, the people who developed ITS did not share the same global mission as those who developed CTSS and Multics. Despite their strong beliefs that their system was the right way to implement timesharing, the hackers never seemed to care if anyone else adopted their point of view. Where Corbato and Fano enrolled huge corporations in building a timesharing system for the masses as part of a larger effort to change the way companies thought of computing, the hackers ignored the potential global implications of their system. CTSS and Multics then received most of the attention of the general public because they were the systems that would bring computing to the masses; the network of people enabling the development of CTSS and Multics – the researchers who would use it as a computer utility, the managers of corporations that saw their computer market expanding – were people who never intended to study the computer itself. ITS received lesser public interest because it was designed more as a vehicle for studying the computer. ITS was thus not suited for the kind of general application that was the goal of CTSS and later Multics; different social factors drove the technical design of the system. The system design that was so optimized for group collaboration and sharing could never survive in a potentially hostile environment without significant changes to the security implementation. The lack of security is what later killed ITS when systems were being evaluated for their applicability to ARPANET. The mini-computing model adopted by the AI Group had enormous value to the research community, but lost more general audiences because all of the features that made the system ideal for research conflicted with those that would make it a viable computer utility.

As with any engineering project, the social and technological factors influencing the objectives, goals, implementation, and success or failure of the final product are strongly connected and largely inseparable. The emergence of the MIT Artificial Intelligence Laboratory and the parallel story about the development of timesharing embody the intimate relationship of technological influences and social forces, and offer insight into how these pressures combine to drive engineering revolutions.

REFERENCES

"Abridged CTSS Programmers' Guide" in Technical Information Program TIP-TM; 113. Cambridge, MIT, 1968.

Bawden, Alan. "PCLSRing: Keeping Process State Modular". at http://www.inwap.com/pdp10/pclsr.txt

Belzer, Jack. et al. Encyclopedia of Computer Science and Technology. University of Pittsburgh. Pittsburgh, Pennsylvania. Volume 12. Pattern Recognition to Reliability of Computer Systems.

Belzer, Jack. et al., eds. "Project MAC", in Encyclopedia of Computer Science and Technology. vol 12. 1968.

Brown, Sumner. MS Thesis, EE, 1969.

Charles Babbage Institute. Fernando Corbato Interview. at http://www.cbi.umn.edu/oh/display.phtml?id=96

Charles Babbage Institute. Robert Fano Interview. at http://www.cbi.umn.edu/oh/display.phtml?id=99

Christiansen, Clayton M., The Innovator's Dilemma. New York: HarperBusiness, 1997.

Corbato, F.J. "An Experimental Time-Sharing System". AFIPS Conference Proceedings 21, 1962 SJCC.

Corbato, F.J., and C.T. Clingen. "A Managerial View of the Multics System Development". Conference on Research Directions in Software Technology, Providence, Rhode Island, October 10-12, 1977. at http://www.multicians.org/managerial.html

Corbato, F.J., and V.A.Vyssotsky. "Introduction and Overview of the Multics System". Proceedings of the Fall Joint Computer Conference, Las Vegas, Nevada, November 30, 1965. at http://www.multicians.org/fjcc1.html

Corbato, Fernando. Interview conducted by Craig Music, Kara Sprague, and Rebekah Wahba. November 26, 2001.

Crisman, P.A., ed. "The Compatible Time-Sharing System: A Programmer's Guide 2nd Edition". 1965.

David Jr., E.E., and R.M. Fano."Some Thoughts About the Social Implications of Accessible Computing". Proceedings of the Fall Joint Computer Conference. 1965. at http://www.multicians.org/fjcc6.html

DARPA History: http://www.darpa.mil

DEC Corporation Fan site: http://www.36bit.org

Dennis, Jack. Interview conducted by Kara Sprague and Rebekah Wahba. November 29, 2001.

Eastlake, D., R. Greenblatt, J. Holloway, T. Knight, and S. Nelson, "ITS 1.5 Reference manual". Cambridge, MIT, July 1969.

Fano, Robert. Interview conducted by Craig Music and Kara Sprague. November 29, 2001.

FOLDOC entry on ITS: http://wombat.doc.ic.ac.uk/foldoc/foldoc.cgi?ITS

Garfinkel, Simson L., Architects of the Information Society. Cambridge: The MIT Press, 1999.

Gentry, Michael Lee. MS Thesis, Nuclear, 1966.

Grochow, Jerrold. MS Thesis, EE, 1968.

Hacker's Dictionary: ITS. http://foldoc.doc.ic.ac.uk/foldoc/foldoc.cgi?ITS

Harrenstein, Kenneth. BS Thesis, EE, 1976.

ITS & PDP-10 terminology/folklore: http://metalab.unc.edu/pub/academic/computer-science/history/pdp-10/docs/

ITS History: http://bespin.org/~tcn/pub/hack/its.html

J.C.R. Licklider Papers. MC 499. Institute Archives and Special Collections, MIT Libraries, Cambridge, Massachusetts.

Joe Smith's PDP-10 page: http://www.inwap.com/pdp10/index.shtml

Jones, Malcolm. PhD Thesis, Management, 1967.

Karl Wildes Papers. MC 322. Institute Archives and Special Collections, MIT Libraries, Cambridge, Massachusetts.

Knight, Tom. Interview conducted by Craig Music, Kara Sprague, and Rebekah Wahba. November 26, 2001.

Latour, Bruno, Science in Action. Cambridge: Harvard University Press, 1987.

Levy, Steven, Hackers. New York: Penguin Books, 1994.

Lucas, Henry. "An Online User Information Facility for Multics Time Sharing System" in Project MAC Technical Memorandum; TM-348. Cambridge, MIT, 1967.

MacKenzie, Donald, Inventing Accuracy. Cambridge: The MIT Press, 1990.

McCarthy, John, Marvin Minksy, N. Rochester, and C.E. Shannon. "A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence". 1955 at http://www-formal.stanford.edu/jmc/history/dartmouth/dartmouth.html

McCarthy, John."Reminiscences on the History of Timesharing" 1983 at http://www-formal.stanford.edu/jmc/history/timesharing/timesharing.html

Minsky, Marvin, and Seymour Papert. "Artificial Intelligence Progress Report". Cambridge, MIT, January 1, 1972.

Minsky, Marvin, Phone Interview conducted by Stefanie Chiou. December 1, 2001.

Minsky, Marvin. Email Interview conducted by Rebekah Wahba. October 25, 2001.

MIT Museum. Photos.

MIT. Laboratory for Computer Science, 1961-1988. AC 268. Institute Archives and Special Collections, MIT Libraries, Cambridge, Massachusetts.

MIT. Office of the President, 1957-1966, Julius Stratton. AC 134. Institute Archives and Special Collections, MIT Libraries, Cambridge, Massachusetts.

Moses, Joel. Interview conducted by Kara Sprague and Rebekah Wahba. November 29, 2001.

Multics: http://www.multicians.org

Norberg, Arthur L, and Judy E. O'Neill, Transforming Computer Technology. Baltimore: The Johns Hopkins University Press, 1996.

Papert, Seymour. "The Artificial Intelligence of Hubert L. Dreyfus: A Budget of Fallacies". Cambridge, MIT, July 1968.

PDP-10 emulators, and sources for ITS: http://bony.umtec.com/pdp10.html

PDP-10 information: http://www.dbit.com/pub/pdp10

Pfluger, Annika. Photos from AI group ca. 1957-1969.

Richard Stallman on ITS: http://www.gnu.org/philosophy/stallman-kth.html

Richard Stallman story: http://www.coyote.org/~jonas/gnu-first.txt

Saltzer, Jerome. "CTSS Technical Notes". Project MAC Memorandum MAC-M-153, Cambridge, MA, July 21, 1964.

Saltzer, Jerome. Email Interview conducted by Rebekah Wahba. November 19, 2001.

Saltzer, Jerome. Interview conducted by Stefanie Chiou, Kara Sprague and Rebekah Wahba. November 29, 2001.

Saltzer, Jerome. PhD Thesis, EE, 1966.

Sekino, Akira. PhD Thesis, EE, 1972.

Space War Game: http://www.wheels.org/spacewar

Sussman, Gerald Jay, and Terry Winograd. "Micro-Planner Reference Manual". Cambridge, MIT, July, 1970.

Sussman, Gerald. Interview conducted by Craig Music, Kara Sprague and Rebekah Wahba. November 15 and 29, 2001.

Turing, Alan. "On Computable Numbers with an Application to the Entscheidungsproblem", The Proceedings of the London Mathematical Society. 1937.

Vintage Computer Festival: http://www.vintage.org

Vleck, Tom Van. Email Interview conducted by Kara Sprague. November 28, 2001.

Waldrop, Mitchell M., The Dream Machine. New York: Viking, 2001.

Wildes, Karl L., and Nilo A. Lindgren. A Century of Electrical Engineering and Computer Science at MIT 1882-1982. Cambridge: MIT Press, 1985.

Winett, Joel Max. Elec. E., EE, 1965.

Winston, Patrick H. "Learning Structural Descriptions from Examples". Cambridge, MIT, September 1970.

Winston, Patrick. Interview conducted by Craig Music and Kara Sprague. November 6, 2001.

Zuga, Brian K. "Tape Archiving Using the Time Capsule File System". June 1995 at http://www.swiss.ai.mit.edu/~boogles/papers/tcfs-thesis/thesis.html