KHP
SERVICES, INC.

POST OFFICE BOX 898837 • CAMP HILL, PA 17089-8837 • (717) 763-3280

DATE: February 7, 1996 (revised 2/9/96)

TO: Fred Heavy
Brenda Smoker

CC: Jack Rutt
Michelle Triponey
Tim Snyder
Scott Thomas
Beth Bachman

FROM: Jeff Szuhay

RE: Recent KHP5 events & a possible solution proposal

We have recently been witnessing the progressive throttling of our Tingley systems performance. Recent events with KHP5 further bring this problem into sharp focus.

This problem could have been avoided, and can be completely avoided if preventative measures are now taken. The solution proposed below fits within our current software methodologies initiative but goes further *now* in that it is intended to deal with existing code running today, not future software development.

I would like to state here my view of the problem in as factual manner as possible; it is not my intent to cast aspersions or pass judgements. By examining the current situation, I hope to provide appropriate expectations of the proposed solution.

The Problem—Where We Are

There are two parts to the problem. The first is the Pick system; the second is our manner of developing and deploying software as we have become a "large" data processing shop.

The Pick System. Pick is an incredibly powerful database and query system which has helped propel us from a small Pick shop to one of the largest Pick shops in the country. Unfortunately, the strengths of Pick in a small setting (few users, small files, simple processes) become its very weaknesses in a large setting, especially with respect to large files. Programming techniques which work wonderfully in a small setting will and, as we have seen, do choke a large system to its death. This is the nature of Pick (as well as other environments) and can only be avoided by ongoing vigilance through training, design walkthroughs[1], and code walkthroughs, among other software engineering methodologies.

Our Development Process. Here, I want to focus on our current process of software development and deployment as I have seen it. Very often software requirements change throughout the coding phase; deadlines are often set to unreasonably short time frames; consequently, little or no attention is given to designing a solution; when the code is delivered, it is given cursory tests without any general review or concerns for system performance. It is enough that our programmers meet their deadlines let alone consider better coding techniques.

Furthermore, there is a climate where "if its not billable, don't do it." This fosters a climate of zero code maintenance (and little consideration for maintaining code) and rewards sloppy or haphazard coding.

The Problem—Where We Are Going

Several very positive initiatives are underway—migration to UniData, and implementing software quality assurance methodologies. Each of these will be examined as to its long-term efficacy.

The UniData migration. This may provide some relief in the short term (after an intense and painful conversion) but, again, with a lack of large-systems thinking and without review of implemented programs, it seems likely, even the UniData system will become throttled and suffer a similar fate as the KHP5 machine.[2] The adage that comes to mind here is "work expands to fill all the available time / CPU cycles."

Software Quality Assurance. This holds the greatest long-term promise but it will also require an intensive implementation period and will only apply to new development in its nearest term. Existing processes will likely be unaffected until they are modified or eliminated, if at all.

Therefore, for the near-term, neither of these addresses the existing crisis, nor truly prevents such crisis in future.

1. I use the term "walkthrough" in the sense of a peer review / approval stage. See Edward Yourdon's appendix on walkthroughs in *Modern Structured Analysis* for a broad description and application.
2. This could happen within a 6 month to 1 year time frame.

One Proposed Solution—Autonomous Code SWAT Team

I would like to propose that a Code SWAT Team, for lack of any better term, be formed to address coding practices which throttle system peformance. Such a team must needs be autonomous yet have authority to get necessary changes made. The team will work in concert with the systems resource team, the application programmers, the host operating systems team, and the business analysts.

Objective. The primary objective of the SWAT team is to measurably improve system performance *ceteris paribus* within a 3- or 6-month time frame. When the team is no longer capable of proving its value—no further system performance can be gained via coding practices (as opposed to design practices or business process improvements)—the team should be disbanded and reassigned.

Focus. The focus of the team is to augment existing operations efforts. Whereas other teams look at Pick processes from the Unix level, this team's focus will be on getting inside the Pick BASIC code and making coding changes. Whenever a Pick process is found to be consuming inordinate amounts of system resources, our team will go into the code; other teams will deal with macro-system considerations.

The team will not address design or business process issues. It will be assumed these are the bailiwick and focus of KHPS's overall software quality assurance and software engineering initiatives (currently under Beth Bachman with Scott Tucker providing practical approaches).

As software engineering practices are implemented, it is anticipated that members of this team will be a part of design reviews prior to coding and peer code reviews prior to implementation.

Resources. The team should consist of no more than 6 full-time staff. Optimally, some staff may be assigned on an either part-time basis or an ad hoc basis. Their capabilities should include advanced Pick BASIC programming experience and a systems-level knowledge or awareness. It is expected that the heaviest demand (and greatest return on effort/investment) will be within the first 6 months and taper off thereafter.

At first, focused thinkers will be preferred over generalists or broad thinkers—but that emphasis may change over time.

A minimum staff for this team would be a Team Leader and two full-time assignments. This could be augmented with variable, part-time assignments.

Based on the above considerations, I would request the following staff:[3]

Full-time assignments:	Jeff Szuhay	Team Leader
	Jim Galliher	Pick & systems skills
	Mark Ott	Pick & applications skills
	Jim Boyles	UNIX & performance skills
Part-time assignments:	Tim Snyder	Pick & systems skills
	Paul McHugh	Troubleshooting skills
	Ken Osmolenski	Pick & application skills
	Terry Posten	Pick & Application skills
	Keith Tingley	Overall skills—all areas
	Joe Keller	Beginning Programmer

It should be noted that emphasis is given to a team with a variety of skills and perspectives. The staff recommended have demonstrated their abilities and desire to succeed.

Approach. The basic approach will be to examine existing code for known "deadly" coding practices, make necessary changes with the approval of pertinent business analysists and applications programmers, and provide ad hoc training and design recommendation where possible and appropriate.

It is expected that Scott Thomas's group and Tim Snyder's group will provide immediate focus as the system becomes throttled with running processes. Also, Michelle Triponey's group will provide longer term focus as client demands are planned and implemented.

It cannot be emphasized enough that the SWAT team should be both autonomous and have the authority to overcome any natural resistance from programming staff, business analysists, or middle management.

Time Frame. Immediate. We are not currently examining code for efficiency. Any steps in this direction are highly likely to provide immediate benefit.

Furthermore, we know what to do now; we have never had a team or the corporate commitment to do it (refer to Tim Holland's repeated recommendations over the years).

3. These represent "first thoughts." I would welcome other recommendations.