Profile of NASA Software Engineering: Lessons Learned from Building the Baseline

Eighteenth Annual Software Engineering Workshop
2 December 1993
Goddard Space Flight Center

Dr. Dana Hall
Science Applications International Corporation
and
Frank McGarry
NASA Goddard Space Flight Center

It is critically important in any improvement activity to first understand the organization's current status, strengths, and weaknesses and, only after that understanding is achieved, examine and implement promising improvements. This fundamental rule is certainly true for an organization seeking to further its software viability and effectiveness. This paper addresses the role of the organizational process baseline in a software improvement effort and the lessons we learned assembling such an understanding for NASA overall and for the NASA Goddard Space Flight Center in particular. We discuss important, core data that must be captured and contrast that with our experience in actually finding such information. Our baselining efforts have evolved into a set of data gathering, analysis, and cross-checking techniques and information presentation formats that may prove useful to others seeking to establish similar baselines for their organization.

Role of the Baseline in Process Improvement

We use the term "baseline" to mean a relatively detailed understanding of the software engineering products, processes, and environment characteristic of an organization, large or small, in a given period of time. It is a snapshot of current product attributes and of present software engineering processes and the environment within which those processes operate. The fundamental objective is to gain understanding and not to judge that the way the organization performs its software development, maintenance, management, and assurance is right or wrong. This understanding is then used in two principal ways; first,

to help identify and define potential improvements and, second, to serve as a reference against which future comparisons are made as candidate improvements are prototyped and implemented.

Establishing the baseline is the mandatory first step of any process improvement program. Determining the organization's software and software engineering characteristics before proposing and trying an improvement requires discipline. It is reasonably analogous to the discipline required to first understand a software problem's requirements and design before jumping in to write code. Although some random "improvements" might prove correct, experience has repeatedly shown that most are off the mark and are almost always short lived, frustrating, and wasteful of people's interest and resources. Thus, it is very important that the time and energy be taken to first gain insight and to understand the what's, how's, and why's of an organization's way of doing software business.

The understanding step provides the foundation for all of the process improvement program. The figure on page 4 illustrates the iterative and chronological relationship between the three fundamental steps of the basic process improvement paradigm. The Figure shows that gaining understanding precedes and then parallels the assessing and packaging steps. Examples of the types of insight comprising the understanding step are shown and will be further discussed below. Note that the understanding process is never completed. It continues on through repeated cycles of update as well as probing into lower levels of detail when and as needed to facilitate focused assessing and packaging. The assessing step uses the insight gained from the understanding activities to identify and define focused improvements that appear to be beneficial and cost-effective. The assessing activity includes prototyping and experimentation. Examples of such experiments are trying an improved inspections process or alternate testing technique. Those improvements that do prove helpful are then packaged as policies, standards, or guidebooks for promulgation back into the organization. Over time and with continued attention, the modifications become a routine part of the organization's software culture and the fundamental software engineering baseline of the organization thus will have advanced.

The figure on page 4 also helps to show that policies, standards, and guidebooks evolve from hands-on experience and usage. As a result of actual implementation in the culture of the organization, the policies, standards, and guidebooks serve as a means for communicating and helping to achieve greater uniformity. The people in the organization own and recognize the "rules" as simply their way of doing business. Experience has

shown that this bottoms-up process is much more effective and useful than is adopting or tailoring a process from another organization or from some top-down mandate.

Core Data You Want to Capture

The organization's software characteristics baseline consists of four categories of information. These categories are:

1. Insight about the organization's application domain(s)
2. Characteristics of the end items; i.e., the products and/or services the organization provides
3. Attributes of the process(es) the organization uses
4. Insight about the organization's environment; i.e., its supporting tools and computing and networking infrastructure.

The figure on page 6 presents the core data that comprise these four information categories. This is the basic data you want to find in order to achieve a first order understanding of the organization and its software practices. Most fundamentally, insight is required about what the organization's application domain or domains are. In other words, what does the organization do, what people and budget resources does it have, and where is the organization trying to go (what are its goals?). Software development and maintenance often are only a part of its purpose so insight must be gained about the organization's overall roles and its software work as a subset of those roles. Further, since many organizations perform work in more than one domain, understanding must be gained about the allocation of resources across those domains.

Using the knowledge of the organization's application domain(s) as a foundation, insight is then gathered about the its products, processes, and computing environment.

Product insight is most readily quantifiable. The amount of software under development and the amount being maintained, languages in use, and error characteristics of the operational software are examples of key product attributes. As illustrated on page 7, we learned in our baselining activities at the NASA Goddard Space Flight Center (GSFC) that of the total civil service and support contractor population (a community of some 12,000 people), roughly one third spend the majority of their time doing software engineering-related work. By "software engineering-related", we mean performing one or more of the

functions of software management, requirements definition and analysis, design, coding, testing, configuration management, and quality assurance. That community is responsible for the maintenance of at least 43 million source lines of presently operational code. We have found that across NASA all software activity groups into one of six major application domains: flight/embedded software, mission ground support software, general support software, science processing software, administrative software, or simply (for lack of a better title) research software. The distribution of existing operational software at the GSFC into the major NASA software application domains is shown in the figure on page 8. By way of definition, mission ground support software is the ground software necessary for the preparation and conduct of a space flight mission. Examples are command management software and software for determining vehicle orbital position. General support software includes engineering models, simulations, tools, and similar operational software. While pie charts are a useful analysis aid, other formats are also helpful. The graphic on page 9 uses a histogram format to show some of our baseline about software language preferences at the GSFC. This particular figure compares the language characteristics of currently operational software and the preferences of the developers of new software. (Neither the GSFC overall nor any of the organizations that comprise the space center have mandated or advocated a software language or set of languages. Each developer or project typically selects the language they prefer.) It is interesting to note the sharp decrease in FORTRAN use and the significant increase in the preference for C and C++. (Our baselining did not, unfortunately, distinguish between C and C++. We may look more specifically into the use of those two languages in the near future.) Ada use has grown at GSFC, but not to a prominent amount.

An organization's process characteristics may prove more difficult to determine. The availability of such insight in any reasonably quantified way is, in our experience, a good indicator of the organization's software engineering maturity. The managers of an organization with immature software engineering practices are not able to easily describe and prove usage of their policies and standards, the usual allocation of resources by software phase, tools used and the usefulness to the organization of those tools, or other key process attributes such as frequency and characteristics of reviews, staff training practices, and project software configuration control activities. An example of understanding an organization's process characteristics is shown in the schematic of page 10. This graphic portrays the usual GSFC software project's consumption of resources by each of the major phases that make up the software development process. This figure is presented here simply as an example of process insight, but it is also an interesting figure to

briefly discuss. We found this allocation of resources to be roughly constant at GSFC regardless of activity development approach or size. While we have, for simplicity of presentation, mapped resource usage into a classical, one pass software development model, many projects at GSFC (especially the smaller ones) develop their software through an iterative, build-it and try-it approach. Almost invariably, these one person and small team efforts claimed not be following any particular development model and certainly not to be doing anything as formal as "requirements definition", but in reality, we observed that their work processes do cycle through the basic four phases, albeit in an informal, less structured way. So, our baselining indicates that the resource usage summarized by the schematic on page 10 is fairly typical of most software work at the GSFC.

A fundamental rule associated with establishing an organization's software engineering practices baseline is to not judge during the baselining process whether an observed practice is good or weak. The objective is simply to observe and record. One of several classical software engineering rules of thumb, for example, says that the front end requirements and design processes should receive 40 percent of the development resource budget while coding receives 20 percent, and testing the remaining 40 percent. GSFC software work varies some from that guidance. We do not at this stage argue or even examine whether some alternate resource pattern might be more effective for the typical GSFC software effort. However, comparisons such as this help to highlight further exploration to be done during the subsequent Assessing phase.

The fourth area of required insight concerns the organization's computing environment. This part of the baselining focuses on attributes such as the types of computers available, how networked the organization is, and how integrated are its software tools. An objective is to measure the organization's computing environment relative to the current state of practice generally in place across the software industry and to identify and characterize constraints that limit the organization's ability to continually or periodically upgrade. Aside from budgetary pressures, a constraint in the GSFC environment, for example, is the large quantity of currently operational software that must be maintained. Taking care of that installed base of some 43 million source lines of code inhibits the modernization of much of the computing infrastructure. We noted, as a consequence, continued reliance on centralized processing and limited introduction of more recent advances such as client-server architectures and powerful desktop computers. Understanding the organization's constraints with respect to its computing environment helps to set the practical context within which incremental improvements are possible.

What Data Can Be Captured?

While a core of data is critical as the foundation for an organization's software engineering improvement baseline, the reality is that often much of even that core data is not available. This is especially the case in an organization whose software engineering practices are relatively immature. Immature is meant as a descriptor of organizations where software engineering practices are largely driven by individual preferences, where little or no organized measurement is performed, and where there is little uniformity and sharing across the organization. The Software Engineering Institute at Carnegie Mellon University labels such organizations as "level 1" in its five level capability maturity model.

Establishing a baseline is an incremental process in and of itself. As the figure on page 4 emphasized, understanding begins as the first step and then is a process that continues in parallel as the organization experiments with and implements focused improvements. The data gathering process is one of iteratively gaining sufficient insight to identify and define promising improvements. Detailed accuracy and depth are not necessarily needed at least initially for the organization-wide baseline. As candidate improvement areas are identified, however, more indepth probing will usually help to understand weaknesses and to shape the nature of candidate changes. More will be said a little later in this paper about the balance between resources put into the baselining process and the level of depth and accuracy of the baseline.

In our GSFC and NASA-wide baselining work, we have been reasonably successful gathering insight about software languages in use, budgets, and quantities of software (as measured by lines of code and people involved). We estimate that our results for these types of measures are accurate within 25 percent of the true amounts. Twenty five percent is admittedly a wide margin, but it is adequate for our software process improvement needs at this early stage in the NASA Software Engineering Program. We have not been as successful identifying other less tangible core data. Examples of such data include effort distribution by phase, the operational lifetime (longevity) of software, error statistics of any kind, productivity measures, and the amount of resources typically invested in the key "overhead" functions of software quality assurance, configuration management, documentation, and project management. As discussed later in this paper, we believe that we have directly or indirectly gathered data from about ten percent of the GSFC software community. Very few of the managers and staff with whom we interacted had any

quantified data pertaining to these less tangible measures. Most could offer only qualitative guesses. While such insights are probably better than no insight at all, we put a wide margin of error of 50 percent on that subset of our characteristics baseline. The point of this discussion is to emphasize that although a core family of process attributes are important to your understanding baseline, practical circumstances may dictate that you settle, at least in the early stages of an improvement program, for approximations and opinions for some of the desired data.

Techniques for Establishing the Baseline

We have found over the past year and six months that a combination of four methods works best to gather, cross-check, and understand the software domains, products, processes, and environment characteristic of a subject organization. The four techniques that comprise our integrated data gathering mechanism consist of administered survey vehicles, informal roundtables, review of selected project data and documentation, and one-on-one interviews. These techniques reinforce each other. Our experience indicates that all four are necessary in order to truly understand the what, how, and why of an organization's software business.

Surveys are a key instrument. We developed, tested, and placed under configuration control a comprehensive eight page survey and a single page, special subset. The single page version was used in two ways. One application was to help introduce our purpose to senior management, garner their support and approval, and elicit their insight about the software engineering process in their organization. The other way we used the single page version was as a verification mechanism with various individuals throughout the organization that had not been interviewed using the longer survey.

The eight page, main survey was widely applied. We found the most effective administration method to be a technique similar to that of a census-taker. At a pre-arranged meeting time and location, our data gatherer met with the single or occasionally several respondents. After introductions, a short explanation would be given of the NASA Software Engineering Program and the role of the characteristics baselining activity within that Program. The data gathering meeting would then proceed in the style of a question and answer session using the survey as the central script. A simple "don't know" or "not available" was entered as the response for those questions where the respondent could not

easily answer. We had structured our questions so that a knowledgeable respondent would not need to invest time researching and compiling answers. Typically, the data gathering session consumed about 1.5 hours, although related discussions would sometimes extend the duration. This method of meeting with the respondent and walking down through the survey not only used minimal time, but also assured us of data return and data interpretation consistency. Since we usually used only one and occasionally two data gatherers, we were able to maintain a relatively consistent interpretation and level of detail. Early in our baselining, we tried and discarded easier techniques that relied upon survey mailouts and telephone calls.

Prototyping of the survey mechanisms proved very important. Such testing helped identify confusing questions, inconsistent definitions, and to polish our gathering techniques. We found, for example, that entries for descriptive data introduced too much variability and thus complicated our data reduction job. A more effective approach was to only use questions with definitive answers such as yes/no or response ranges (for example, <35%, 35-70%, or >70%).

The roundtable sessions were used to help check the insights gathered from the administered surveys and to gather more subjective opinions and advice. The roundtables were conducted using a structured set of five major questions; specifically:

1. Participant background and experience?
2. Business goals and objectives of the organization?
3. What software process is used?
4. Major strengths and weaknesses in developing software?
5. What could be done to improve the software engineering process in the organization?

Separate roundtables were conducted for managers and technical personnel and for civil servants and support contractors so that each set of participants could speak more freely. As with the survey and interview results, we have taken care to protect the privacy of the responses. In no case have observations been attributed back to individual participants.

While our survey administration method was in reality a one-on-one or occasionally one-on-two interview, we also used the interview technique principally as a means of pursuing insight that appeared, after analysis and comparison, to be contradictory or in some way particularly different from what we were learning was the norm in the organization.

Resource practicalities obviously precluded talking with every member of the software community so we tried to orient our data gathering energy to the major "pockets" of software work. Senior management proved very helpful in pointing to those software intensive groups within the overall organization, but our knowledge of NASA was also key. This brings up another valuable lesson. It is our opinion that the software engineering baselining process can only be done by individuals familiar with the target organization and its culture. That insight has proven highly important to our efforts at interpreting terminology, understanding roles and functions, and interpolating and extrapolating the data samples to represent the overall GSFC and NASA software communities. We do not believe that we would be very effective if we attempted to conduct this critical activity in unfamiliar domains or, conversely, if someone not familiar with NASA tried to conduct a NASA software baselining.

How Many People Should You Talk To?

No easy answer can be provided to this question. The amount of interaction depends upon the variability within the organization of interest and on its software engineering process maturity. The problem of sample size is probably amenable to statistical analysis. We have relied upon our extensive NASA experience base as our primary guidance for determining our sample size. As the graphic on page 13 shows, we sampled approximately ten percent of the GSFC software community and from that sample size, believe we have extracted insight sufficient to both guide our next round of focused improvement thrusts and to serve as a yardstick for future comparison. We cannot judge that ten percent is a proper sample size for improvement endeavors in other organizations, however.

How Much Will the Baseline Cost?

We have invested approximately eighteen person-months in the baselining activities focused on the GSFC software community. Our efforts first concentrated on the largest and most software intensive Directorate at GSFC and then broadened to encompass all of the GSFC, but at a lesser level of detail. As the data on page 14 shows, a cumulative six person-months was used gathering insight using the integrated four method approach previously discussed. This six months was preceded by two person-months of survey development, testing, and refinement. We found the archiving investment to be extremely important. This function helped to maintained order and organization amidst the inflow of

large quantities of data. We estimate that about six person-months have been invested extracting and deriving information and insight from the survey, roundtable, and interview data. This function includes recognizing and dealing with data overlaps and gaps as well as performing analyses and comparisons. A total of two person-months has been expended so far packaging our results into two profile reports (one for the major software Directorate and one for the GSFC overall) and into several briefings.

Page 14 concludes with a short synopsis of our next steps. We are now transitioning from exclusive concentration on just gaining understanding to an effort balanced between continued understanding activities and focused assessments and prototyped incremental improvements. The software engineering baselining performed to date has highlighted several process areas as promising candidates for improvement.

Training and standards are high on that attention list. Several comments in this regard may be helpful. The GSFC training office has an excellent record of responding to managers' requests for specific training classes. Our observations conclude, however, that a more comprehensive software training activity may be cost-effective. We are interested in examining the advantages and problems surrounding an integrated software engineering training program: a curriculum that routinely prepares personnel for upcoming software roles. We also want to explore whether training effectiveness can be improved by expanding the delivery of training from the traditional classroom to include reinforcement mechanisms such as easy access to help and information from each person's office desktop machine.

The standards area apparently requires a considerably different approach from that used within the GSFC to date. (Since we haven't completed our NASA-wide baselining, we can't fully conclude that the same issue applies across all of the Agency, but our insight so far leads us to think that it does.) Several observations are particularly relevant. First, the existence of advocated software engineering processes and supporting standards is very inconsistent. Few organizations that do software work have any recommended approach at all despite the importance of software engineering to their existence and to the credibility of their products. Second, within those few organizations that do have a recommended software process and supporting standards, managers and staff either claim not to know of the recommended approach or freely take broad license to tailor and selectively apply elements of it. Related to this issue is the tendency for civil service personnel to require the contractor community to follow a particular process and standards while they themselves

know very little about it and exercise no similar discipline on their own software activities. There is a distinct lack of ownership by the using community of processes and standards imposed from outside their immediate organization. We think these interrelated issues sum to the conclusion that most current approaches to developing, advocating, and using software process and engineering standards simply do not help and instead actually hinder, frustrate, and waste resources. Since current software standards methods largely don't work, our next steps in this part of the improvement program will be dual thrusts of continued detailed understanding and prototyping of alternate techniques. A very promising overall approach is that represented by the three layer, iterative model previously shown on page 4. This technique basically plays back into the organization the methods and practices the organization itself or a subset of the organization such as a particular project, has found helpful. These "packaged" methods and practices are the organization's processes and standards and since they are based on actual experience within the organization, they are owned and used with much greater effectiveness.

Lessons We Learned

In summary, the understanding activity is a mandatory and continuing element of any organizational software engineering improvement program. We have now completed the initial understanding baseline for software across the 12,000 person GSFC community. Several lessons from that work may be beneficial to others seeking to establish similar baselines for their organization.

A primary lesson is to be objective. The purpose of the understanding activity is to learn and not to judge that current practices are good or bad. As the understanding builds, a change in perspective can occur to identify candidate areas for improvement, but the key is that shift in perspective be based on facts rather than speculation.

It is important that insight be gained from personnel at all levels and roles within the organization. We found that interacting initially with upper management was especially important. Not only could the upper manager orient us to the types of work and the responsibilities of the organization, but we also gained the manager's approval of our efforts which in turn helped gain time and attention from the staff within the organization. Another benefit of starting with upper management was the important aspect of buy-in by the upper manager to the concept of software engineering improvement. This acceptance becomes especially important downstream when the initial understanding is achieved and

the task of defining and experimenting with promising improvements gets underway. Gaining multiple perspectives from personnel throughout the organization does come with some problems though. The chief difficulty is recognizing and resolving overlaps in data and insights.

Lesson 3 on page 15 recommends layering the baselining effort. In other words, gather insight that is truly representative of the way the organization does its software work, but go only into as much depth as is needed for the current stage of the improvement effort. As candidate improvement areas are identified, more indepth investigation can be done concentrated on the aspects of that candidate area. As in any data gathering exercise, it is very easy to become overwhelmed with data and not be able to discern from all the data the useful information. Keeping a carefully organized archive helps, but the real key is to maintain a perspective of "peeling away layers of insight" as is most useful to your stage of improvement.

As the understanding builds, package the insight into some type of communicative medium. We have used both briefings and reports (which we call "software engineering profiles") as convenient repositories to store our insight and facilitate further discussion and progress within the organization. To help these products mature, give members of the organization opportunities to read and comment. This will likely be a challenging activity, however, because your baselining work will have identified weaknesses and problems which the organization may not want to hear or see on paper. Again, the support and commitment of the upper managers to process improvement become important contributors to the success of your efforts.

Finally, we have evolved to a combination of four methods to gather and verify process insight. Surveys work well if administered in person, if thoroughly tested for completeness and consistency, and if conducted by a single or at most very small team of personnel knowledgeable about the kinds of work the organization performs. Roundtables are a means of gathering in more subjective opinion and perspective. Reviewing selected documentation and data and follow-up interviews serve as tools for verifying and clarifying important items in your evolving understanding baseline.

References

1. Software Engineering and Assurance Plan, draft, July 23, 1993, NASA Headquarters Office of Safety and Mission Quality.

2. Profile of Software Within Code 500 at Goddard Space Flight Center, NASA Software Engineering Program, December 1992.

3. Profile of Software at the NASA Goddard Space Flight Center, NASA Software Engineering Program, draft, December 1993.

Profile of NASA Software Engineering: Lessons Learned from Building the Baseline

Dana Hall
Science Applications International Corporation

Frank McGarry
NASA/Goddard Space Flight Center

Eighteenth Annual
Software Engineering Workshop
December 2, 1993

Topics

- Role of the Baseline in Software Process Improvement
- Core Data You Want to Capture
- What can be Captured?
- Techniques for Establishing the Baseline
- Lessons We Learned Assembling the GSFC/NASA Baselines

Role of the Baseline in Process Improvement

- Objectives -

1) Establish the Baseline
   - Snapshot present attributes of the software itself (Software Product)
   - Snapshot present software engineering practices (Software Process)

Basic objective is to <u>understand</u>; not to judge right or wrong

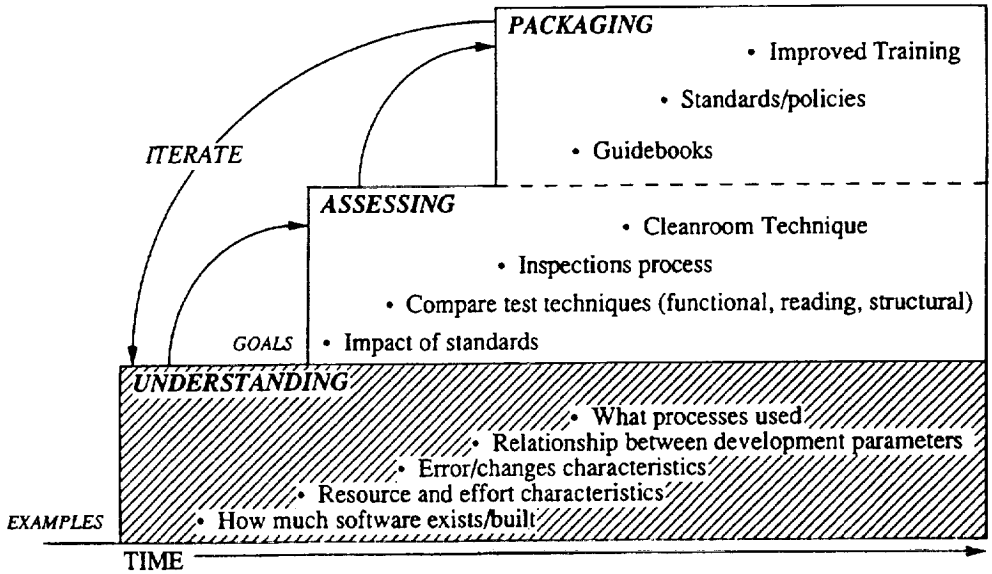
2) Baseline will be used to:

   - Identify and define potential process improvements
   - Make future comparisons to measure progress

Baselining Activity is Mandatory First Step of any Process Improvement Program

(3)

Evolving to an Effective "Process Improvement" Environment



Key Ongoing Step is to Understand

(4)

Most Significant Baseline Data

Core Data you Want to Capture

- Insight about the Application Domain
  "Characteristics of the Problem Addressed"

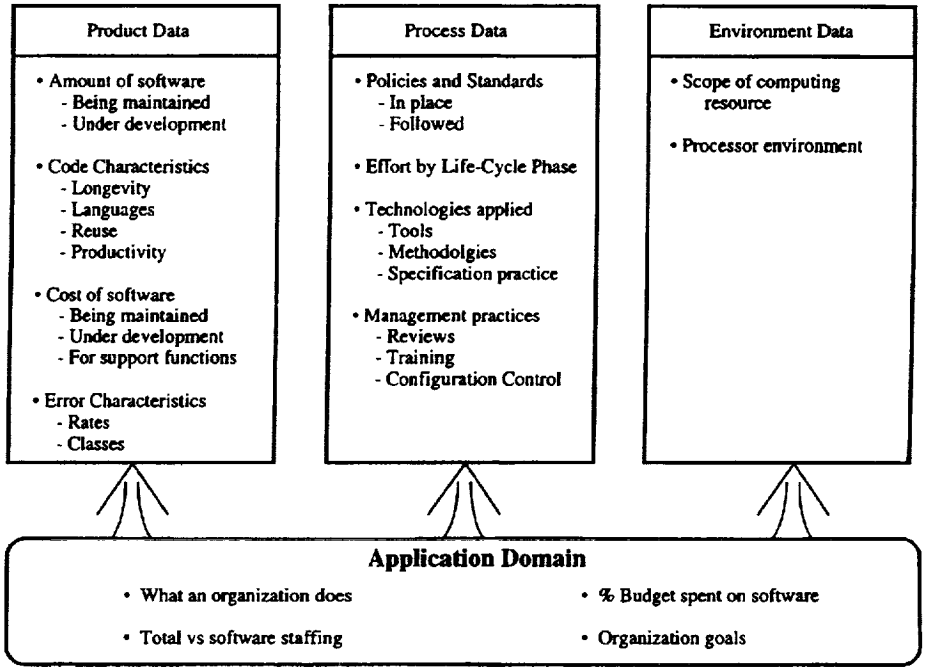
- Product Data
  "End Item Characteristics"

- Process Data
  "How is the End Item Developed and Maintained"

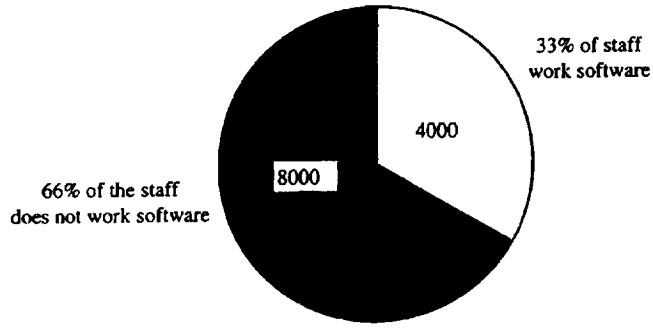
- Environment Insight
  "Supporting Tools and Infrastructure"

(5)

Core Data you Want to Capture



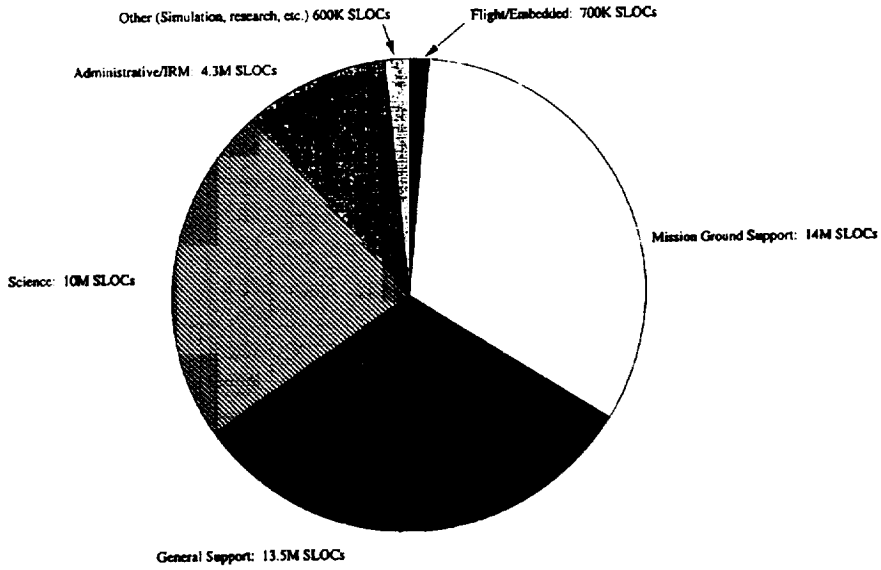
(6)



Total Software Staffing - 12,000
(civil servants & support contractors)

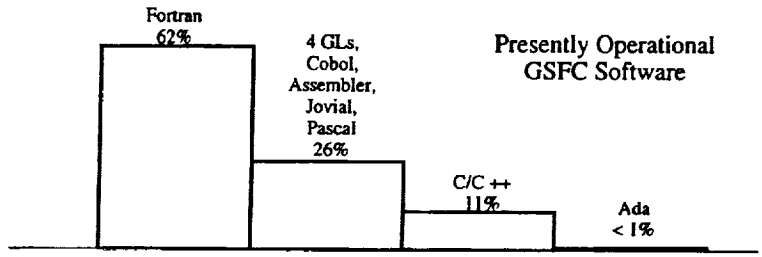
Percentage of Staff Devoted to Software

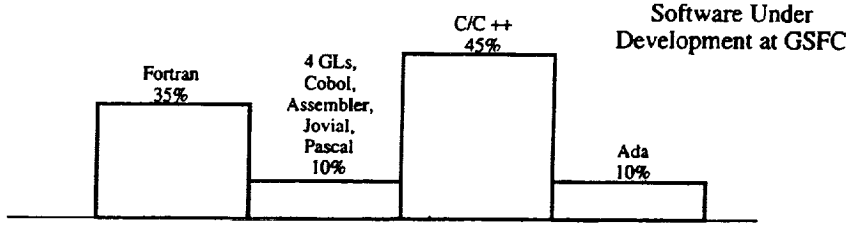
(7)



GSFC presently has about 43 million source lines of code

(8)

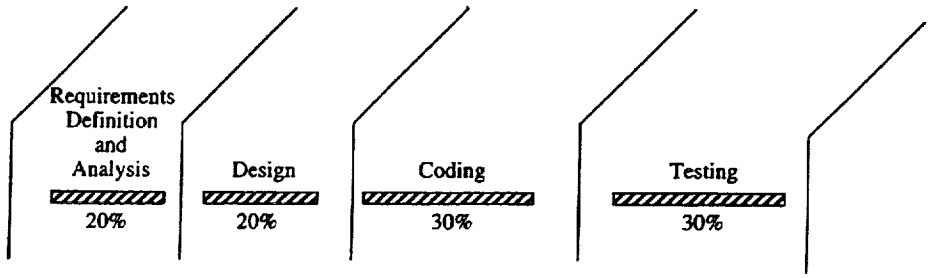




Language Preferences

(9)

Resource Consumption by Development Phase Typically at GSFC



(10)

What can be Captured?

- Conflicts will exist between data you want to capture and what is available/quantifiable

> Data availability is indicator of relative process maturity of the organization

- Accuracy takes long time and many projects (large, overlapping sample size)
- Data you probably can collect:

  Languages
  Budgets (cost)
  Amounts of software

  This data may be accurate ± 25%

- Less tangible data (depending upon organization's process maturity)

  Effort distribution by phase
  Software longevity
  Error statistics
  Productivity
  Investment in "overhead" functions (QA, CM, documentation, management)

  Our experience is accuracy ± 50%

(11)

Techniques for Establishing the Baseline

- Apply combination of four methods:

  - Administered surveys
  - Informal roundtables
  - Data and documentation review
  - One-on-one interviews

- Survey advice

  1) Must prototype/test the survey instrument
  2) Avoid descriptive entries; Make all responses quantities or checkmarks
  3) Use directed sampling

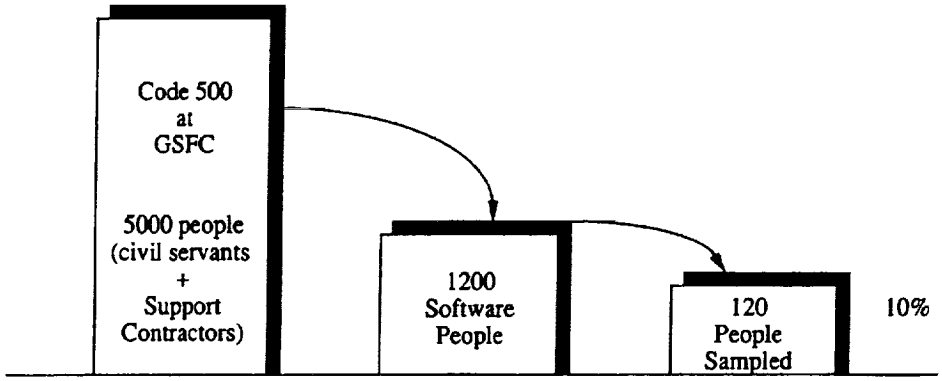
     -- Start with senior managers:
        * organization overview
        * awareness of your activities
        * pointers to "software pockets"
     -- Sample the pockets
     -- Cross-verify

  4) Only one data gatherer or small team (max of 3 people)

  5) Data gatherer(s) must know the organization

(12)

How Many People Should You Talk To?



Sample Size depends upon:

- Organization Uniformity/Heterogeneity
- How many software pockets (Approximately 20 "pockets" in Code 500)

(13)

How Much Will the Baseline Cost?

GSFC Baseline Experience

Survey Development/Testing:	2
Data Gathering (4 methods):	6
Archiving:	2
Data Analysis & Info Extraction:	6
Packaging:	2
	18 person-months

Next Steps: Focus on Most Promising Improvement Areas
1. Training
2. Helpful Standards

Assess/Experiment and Package

(14)

Lessons We Learned

1. Be Objective ⟶ Learn, Don't Qualify

2. Gather Perspective (Cross verify) ⟶
   - Senior Management
   - Lower Management
   - Developers
   - Testers
   - Quality Assurance

3. Layer your Baselining ⟶ Only go as deep as you need

4. Give the Organization Review Opportunity (but don't compromise your findings)

5. Use Combination of Methods:
   - Administered Surveys
   - Roundtables
   - Data Review
   - Interviews

(15)