



*National
Reconnaissance
Office*

Elephant Bungee Jumping

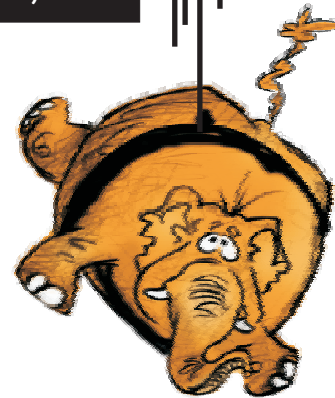
A Software Project Manager's Survival Guide



The downs & (hopefully ups of acquiring software.
For the manager smart enough to know what they don't . . .
(by authors smart enough to never admit to making the same mistake twice)

Authors

Dr. Robert Semelsberger
Mr. Emmett Rigsbee
Mr. August Neitzel
Mr. Robert (SAM) Johnson



Elephant Bungee Jumping

A Software Project Manager's Survival Guide



Loosely Interpreted by

Gary Zelinski and Peggy Poindexter

Illustrated by

Brian Hoard

Layout Artist

Troy Ruda

Contents



| | |
|--------------------|---|
| Introduction | v |
|--------------------|---|

Chapter One Getting Started

| | |
|---|---|
| 1. The Dating Game | 2 |
| Understanding your users and managing their expectations | |
| 2. Easier Than Teaching Pigs to Fly | 4 |
| Selecting and getting the most from your development team | |

Chapter Two Taking Charge

| | |
|---|----|
| 3. No Wimps Allowed | 8 |
| Establishing and controlling requirements | |
| 4. Betting Your Career on Blind Faith | 10 |
| Identifying and controlling risk | |
| 5. A Whip, a Chair . . . and a Bag of Money | 12 |
| Managing cost and schedule | |

| | |
|---|----|
| 6. Help . . . I've Fallen and I Can't Get Up | 14 |
| Using configuration management to control your program's baseline | |
| 7. Metrics Mania | 16 |
| Making decisions based on performance measures | |

Chapter Three Speed Bumps & Potholes

| | |
|---|----|
| 8. A Poke in the Eye with a Sharp Stick | 20 |
| Making modifications to commercial software . . . NOT! | |
| 9. Avoiding Diseases that Are Fun to Catch | 22 |
| Incorporating commercial products into your development | |
| 10. Cutting Edge and Bloody Mistakes | 24 |
| Avoiding unproven technologies in development | |
| 11. Move Over, You're Obsolete | 26 |
| Hardware and operating system upgrades | |
| 12. Reheated Spaghetti Is Still Bad Code | 28 |
| Incorporating re-use in your development | |

Elephant Bungee Jumping

| | |
|---|----|
| 13. No Free Lunch | 30 |
| Using automatic code generators to cut development time | |
| 14. You'll Have Nobody to Blame but Yourself | 32 |
| The pitfalls of Government - furnished equipment | |
| 15. Self-Eating Watermelons | 34 |
| Using multiple development contractors | |

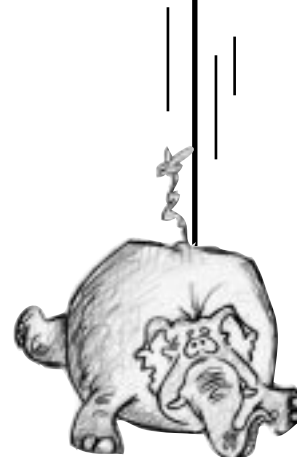
Chapter Four

Finishing Touches

| | |
|---|----|
| 16. Nothing Short of Complete Victory | 38 |
| Transitioning everything to operations | |
| 17. When the Air Tastes Like Water, You Know You're Drowning | 40 |
| When to go for help | |

References

43



The ideas and findings in this guide should not be construed as official NRO positions; it is published in the interest of technical information exchange.

Questions and/or comments should be directed to:

Software Guide
National Reconnaissance Office
14675 Lee Rd
Chantilly, VA 20151-1715

e-mail inquiries to:
SoftwareGuide@nro.mil

Introduction

A *Project Manager's Survival Guide* discusses software acquisition management lessons learned by a few “seasoned” program managers at the National Reconnaissance Office (NRO). It addresses a concern of the NRO’s Deputy Director that many new and innovative management practices are employed across the organization with little or no means to share good ideas or to benefit from the experience of others. Because the software acquisition landscape is littered with numerous technical manuals, the format provides an overview geared to those managers lucky enough to have the challenging responsibility for acquiring new software. It is written primarily for NRO project managers executing their duties as a Contracting Officer’s Technical Representative (COTR). Contracting officers, systems engineers, and other managers involved in software acquisition will also find the guide useful.

The authors and interpreters of this guide drew from personal experiences as well as from inputs gathered throughout the NRO. This guide is not inclusive but presents a short list of software acquisition universal truths based on lessons learned, and often re-learned, on numerous large-scale development efforts over the course of many years.

The discussion of each universal truth is broken down into three sections identifying, explaining, and emphasizing what every successful software project manager should understand. The format provides rationale on why this “truth” was selected and why we believe it is important. Next, the benefits of the truth are highlighted. The third section discusses specific ways that you can emphasize this universal truth within your own program. The final chapter provides a list of resources to help increase your knowledge in a particular area. The suggested resources are some of the more beneficial sources we’ve found. Neither these resources nor this survival guide, however, are meant to replace the management practices or controlling documentation of your organization.

Hopefully, you will find this guide useful and informative as you embark on what can be one of your most challenging and rewarding assignments.

Chapter One

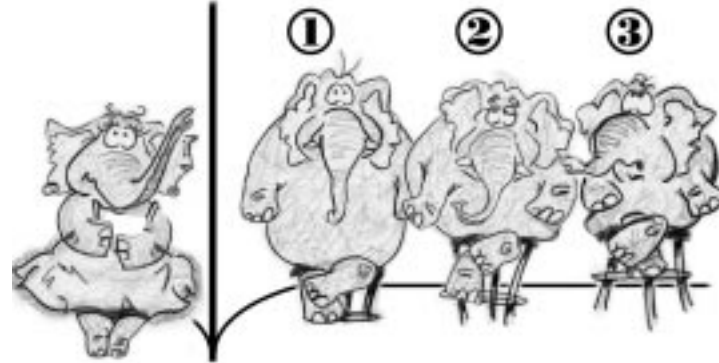
Getting Started



1 The Dating Game

Understanding Your User & Managing Their Expectations

Why This Is Important



Involving users of the system in the acquisition from the beginning provides a greater likelihood of mission success. Their involvement is critical to understanding both the requirements and the operational environment in which the system must work. But be careful that users' involvement does not lead to runaway costs; it's all too easy for users to ask for the moon when they don't have to pay for it!

In order to most effectively implement a system the user needs, take time to understand how your users create and add value to their "mission." Ensure that your project enhances their ability to perform and that they buy-in to

what the project will enable them to do. Beware of users focused on the "what was" – keep them focused on the "what must be."

The longer the development cycle, the more likely it is that the users will reject the system when delivered. Remember, you are trying to hit a moving target; the user base changes and user expectations are constantly remolded by their experiences with commercial products. The situation is exacerbated when you are developing a replacement system because your antecedent will continue to evolve long after you must freeze your baseline.

Benefits

- Greater likelihood that the program will match the needs of the user community.
- Greater likelihood of stabilizing requirements early in the development cycle.
- Greater likelihood that performance trades will have the least impact on the user mission.
- Greater acceptance when performance trades are required due to cost and schedule constraints.
- Update the CONOPS as your user environment changes. Demonstrate your commitment to the user by controlling the CONOPS as you control requirements.
- Map the CONOPS to your top-level requirement specifications.
- Develop and stabilize the user interface early (e.g., prototype) and evolve the functionality to support it.
- Select user representatives carefully and avoid having “too many cooks.” At a minimum, user reps must have recent experience and be respected in their community.

How Best to Emphasize

- Use Joint Application Development (JAD) sessions early in the process to obtain input. JAD sessions are Integrated Product Teams designed to translate user needs into requirements in a manner both users and developers will recognize.
- Develop a concept of operations (CONOPS), written in user language, before beginning the development (including scenarios and use case diagrams, if appropriate).
- Have a process by which the users prioritize their requirements.

Universal Truth #1

Involve users early, regularly, and often
... but in a managed way.

2 Easier Than Teaching Pigs to Fly

Selecting & Getting the Most from Your Development Team

Why This Is Important

Your goal is to achieve tangible, measurable results, not just poetic promises. Past performance on related or similar efforts is still one of the best ways of improving your chances of a successful development. Make sure the winning development contractor has extensive experience with the business domain as well as the proposed development methodology, hardware, software languages, and tools prior to start of contract. Do not let your contract become training for the contractor's team. Focus on understanding your developer's culture. A cowboy attitude can kill you. Look for evidence of mature engineering processes. Ensure that the right expertise for the program is available and

committed. Take time to understand your developers methodology and environment. Make sure they treat their employees well. Don't underestimate how hard it is to hire and retain good people (like your authors). Remember, when you contract for software development, you are contracting for both *people* and *process*.



Benefits

- Resources dedicated to development and not to indirect activities such as training.
- A team focused on the same objectives and understanding the same constraints.
- Greater assurance that the team already has the skills it needs to be successful.



How Best to Emphasize

- Write the RFP carefully and emphasize past performance and relevant technical experience in the award criteria.
 - Specify a dedicated development team in your Statement of Work.
 - Require an oral presentation during source selection by key members of the proposed technical staff.
 - Check every past performance reference you can.
- Plan and budget for the entire effort (e.g., Systems Engineering, Test, IV&V), not just the development phase. Be careful when you include O&M on the same contract and avoid level of effort work.
 - Insist on an accurate schedule and stick to it. If the schedule begins to slip, review the accuracy of the original plan and take corrective action.
 - Insist on a detailed Work Breakdown Structure (WBS). Make sure it is well maintained and correlates to your reported earned value metrics.
 - Don't get caught up in being a zealot for a particular computer, operating system, software language, or development process. Competent technical staffs with relevant experience are key to success.

Universal Truth #2

Watch what they do and not what they say
(this also applies to management, both theirs and yours).

Chapter Two

Taking Charge



3 No Wimps Allowed

Establishing & Controlling Requirements

Why This Is Important

The lack of a firm requirements baseline at the inception of the development effort or a progressive creep in the requirements baseline are major contributors to the failure of many software developments. Requirements often change, but they must be controlled and managed against the impacts to cost and schedule. Late changes to any type of requirement will have costly impacts, especially the further into the development cycle that these changes are made.

Requirements control is essential from the acquisition planning phase through completion. Emphasize both requirements definition and interface definition. Always ask the question: “Where does it say I have to do that?” before accepting new (or even modified) requirements. Custom code can be just as inflexible as hardware. Do not believe

that because it is software, you can accept requirement changes later in the design/development cycle than you can with hardware systems.

Remember, if it is not written down, you don’t get it. Verbal promises don’t last past the first project “speed bump.”



Benefits

- Establishes a clear and concise definition of what must be accomplished (contractual requirements) that is understood by the entire development team.
- Ensures requirements are unambiguous, traceable, verifiable, and documented.
- Defines the scope of the development effort.
- Maintain a mapping of requirements to intermediate products and individual configuration items.
- When you need to change (add, delete, modify) requirements, document the change and track volatility (% change by month).
- Be willing to prioritize your requirements and drop from the bottom of the list when you're forced to make performance trades.

How Best to Emphasize

- If your organization does not have a written policy or procedure for defining and managing requirements, write one. If it does, follow it.
- Designate a group to formally oversee requirements definition and management. A good candidate for this is the test team. Ideally, this group can help ensure requirements are written in a quantifiable way.
- Make sure your requirements management team is trained and uses established plans and procedures.
- Specify performance requirements with future growth in mind.
- **Establish and never lose sight of the commitment to deliver a meaningful and useful product.**



Universal Truth #3

If it's not written down, you don't get it. Verbal promises don't last past the first "speed bump."

4 Betting Your Career On Blind Faith

Identifying & Controlling Risk

Why This Is Important

Software development tends to be one-of-a-kind. This means that, by its very nature, software development is a high-risk endeavor. You will need every imaginable control in place to manage your software development project, and this means actively seeking out risk items for consideration. If you're not thinking about what could go wrong, something surely will.

You need to establish and aggressively use a risk management approach to actively mitigate significant risk elements. You should identify any significant new items/technology/approach as a risk until proven otherwise. Remember, cost and schedules are always risk items until the project is finished.



Benefits

- An active risk management process keeps risk identification and mitigation in the forefront of your thinking.
- A common rating scheme for risk quantification gives a prioritized list of what risks to tackle first (risk = consequence x probability).
- You will not know every program risk yourself. Getting everyone involved in the risk identification process will uncover risks before they become problems that are too difficult or expensive to solve.
- Integrate risk into the development process. Get full buy-in on the process from the team and don't let risk management become pro-forma or an excuse to call a regular meeting.
- For risk identification, use a process that is open to the entire development team.
- Decriminalize the risk identification process. Create incentives for developers to identify risks early through open team communications; develop two-way trust. Don't hammer them on their award fee when they keep you apprised of unknowns. Reward them!

How Best to Emphasize

- Use a formal risk management process.
- Ensure your risk management plan includes both a risk assessment and risk control portion. Update both systematically and frequently – and be flexible.
- Re-evaluate each risk regularly. Attend to each risk mitigation strategy until the risk is reduced or eliminated.

Universal Truth #4

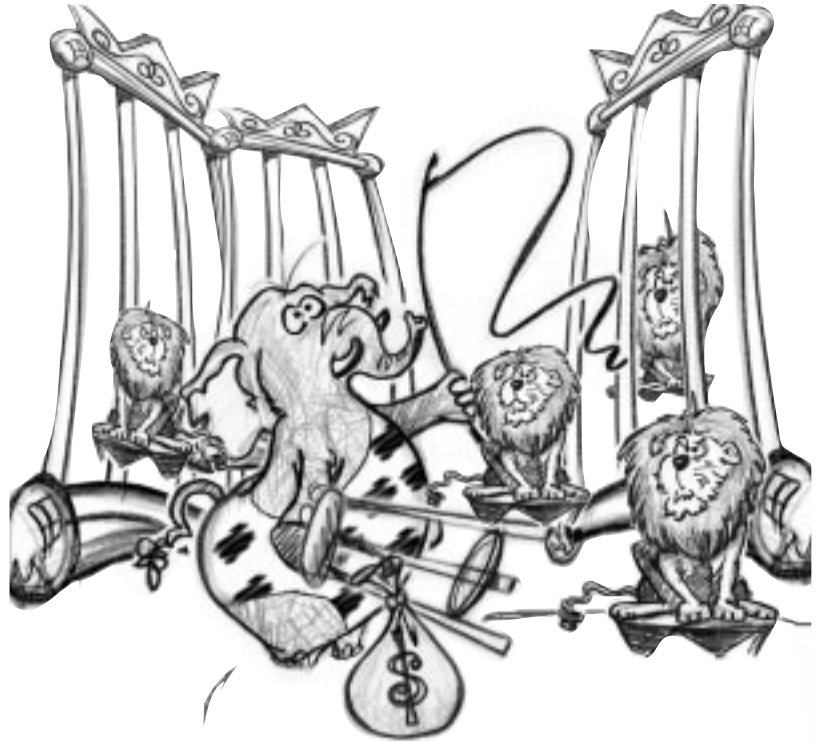
The risks you manage determine the quality you deliver.

5 A Whip, a Chair . . . and a Bag of Money

Managing Cost and Schedule

Why This Is Important

It all starts with a reliable and technically justifiable plan. Remember, if your developer doesn't have a plan that realistically reflects how the system can be built, any measurements made against that plan cannot be trusted to provide meaningful information. Unrealistic and overly optimistic earned value reporting can mask problems and delay corrective actions. Non-development procurements often distort earned value because of the way they are billed and paid. Use other metrics such as staffing profiles, use of overtime, etc., to validate what earned value appears to be telling you. Earned value is only one of several tools available to you, it is not "the" tool.



Benefits

- A good plan and accurate reporting go a long way toward making your job easier. With these two elements in place, your decisions are based on an accurate representation of what work really is being accomplished against a realistically achievable plan.
- You will achieve the most benefit from a good earned value system when you use it to look for and discover defects early.

How Best to Emphasize

- Develop a metrics-based cost and schedule baseline that integrates technical accomplishment into a baseline plan and has as many discretely measured milestones as possible.
 - Conduct an integrated baseline review shortly after contract award.
 - Never write a Level-of-Effort (LOE) contract for a task that requires something to be delivered.
- Progress within software development tasks is notoriously hard to measure. Software elements invariably “hang at the 95% complete stage, so use an earned value system with as close to a 0-100% earned value basis as possible. (0-100% is where earned value is given only when the task is complete.) Never allow a 100-0% earned value method!
 - Identify and correct problems as you go. Establish a peer review process that goes hand-in-hand with the earned value system. The process should be in effect from the requirements analysis phase to the integration and test phase.
 - Think of discovered variances as your program management friend. Decriminalize variance reporting from the beginning for a profoundly positive impact on the overall success of your project.

Universal Truth #5

If you don't know where you're going,
any plan will get you there.

6 Help . . . I've Fallen and I Can't Get Up

Using Configuration Management to Control Your Program's Baseline

Why This Is Important

Configuration Management (CM) is the cornerstone of a solid software development process, especially as the size of the software development grows. Large software systems are so technically complex that without complete and absolute control over the environment, the in-process product, and the requirements baseline, it is virtually impossible to prevent/detect problems until the later stages of development (integration and test).

CM is more than the “golden copy” once development is complete. It is a completely integrated process of identifying and defining the system's product set at established

milestones and controlling the release and change of these items throughout the system lifecycle. It also includes recording and reporting the status of product items and change requests and documenting the completeness and correctness of the product items. To have a truly effective CM program, it must be an in-line process.



Benefits

- Allows accurate tracking of program requirements over the entire lifecycle (analysis, definition, development, and maintenance).
- Allows retrieval of previous working versions and provides an audit trail of modifications.
- Provides tracking of defects by individual configuration items, thereby:
 - Improves overall product quality by identifying problems early.
 - Allows systematic tracking of changes, thereby reducing rework.
 - Prevents uncontrolled, uncoordinated changes and aids immeasurably in problem detection and correction.
- Use review boards to define and enforce CM procedures. Be able to trace defects from the first report to final disposition.
- Allow NO exceptions to established CM procedures. Make sure everyone uses them, including developers, test teams, system engineering, and YOU.
- You have a good CM program if you know exactly what application version, which HW/OS, which database, which defects, and which patches are at each workstation at any given time. If you don't, re-read this universal truth again.
- Encourage frequent working builds during development.

How Best to Emphasize

- Anything that is shared by two or more people should come under configuration control.

Universal Truth #6

Shortcuts in software configuration management will lead to chaos when the "rubber hits the road."

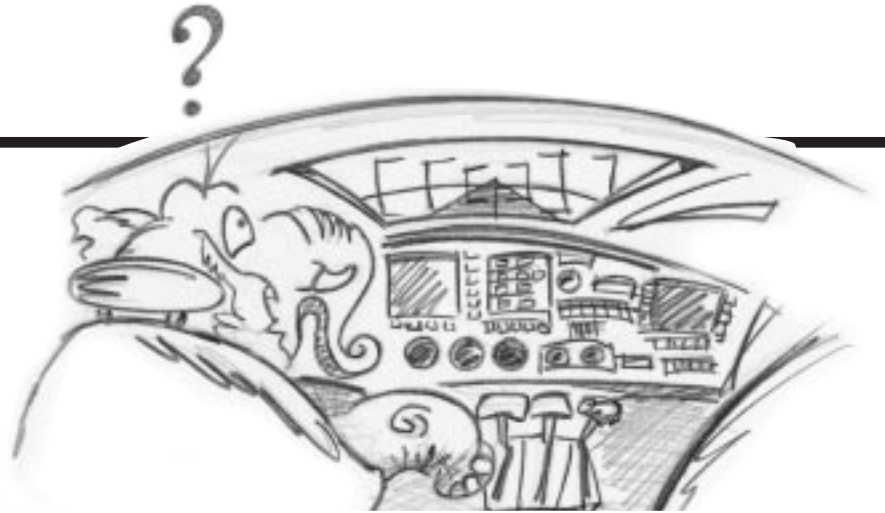
7 Metrics Mania

Making Decisions Based on Performance Measures

Why This Is Important

Performance measures should provide the project manager quantitative insight into the program's health. With this quantifiable insight, you will have a solid foundation to make management decisions. Performance measures should be based on the risk identified and tailored to the mitigation strategy selected. There are literally hundreds of performance measures associated with software development. Many of these are focused on the needs of your development team.

As the Government manager, the decisions you face are similar, but not identical. The focus of the Government manager needs to remain on the impact of cost, schedule, and technical performance. Performance measures allow you greater insight into the root cause of cost, schedule, and technical problems. This insight does not necessarily help quantify the associated impact or specify the action you should take. Know your management's "risk tolerance" and ensure your performance measures help keep you within that threshold.



Benefits

- Objective status on cost, schedule, and technical performance.
- Early problem indication and reporting thresholds.
- Early indications of design or development problems.
- Objective thresholds for making decisions.
- Performance measures must be directly related to the risks you identify and support the actions you take to mitigate those risks.
- Know exactly what decisions you need to make when performance thresholds are exceeded.
- Establish (if your management has not already directed it) reporting thresholds for each measure or combination of measures you select.

How Best to Emphasize

- Develop performance measures using the goal-question-metric method. This method bases performance measures on program objectives and phrases those objectives as quantifiable questions.
- Specify the tracking and analysis of performance measures in the RFP.
- Establish clear performance thresholds for each performance measure.
- Establish acceptable “watch and danger” thresholds for each measure as a function of time. For example, requirements volatility should reach zero by the Preliminary Design Review (PDR).
- If you don’t know what decisions a measure supports – **THROW IT OUT!**

Universal Truth #7

Base your decisions on performance measures.
If it’s not measured, you probably don’t care.

Chapter Three

Speed Bumps & Potholes



8 A Poke in the Eye with a Sharp Stick

Making Modifications to
Commercial Software
... NOT!

Why This Is Important

Making modifications to commercial software causes endless delays, cost overruns, and general mayhem as software packages evolve. Once modified, each evolutionary release causes unforeseen breakage in the software baseline, generating a blizzard of problem reports and setting back development schedules. The pain continues long after the project transitions to operations.



Benefits

- Leave well enough alone.

How Best to Emphasize

- JUST SAY NO!
- This is an absolute prohibition. Violators should be prosecuted to the fullest extent of the law!
- Never pay a COTS vendor to specifically tailor or modify their product for you.
- Corollary: Never allow your developers to obtain source code for any COTS product.



Universal Truth #8

Never allow software developers to modify commercial operating systems or other COTS products.

9 Avoiding Diseases that Are Fun to Catch

Incorporating Commercial Products into Your Development

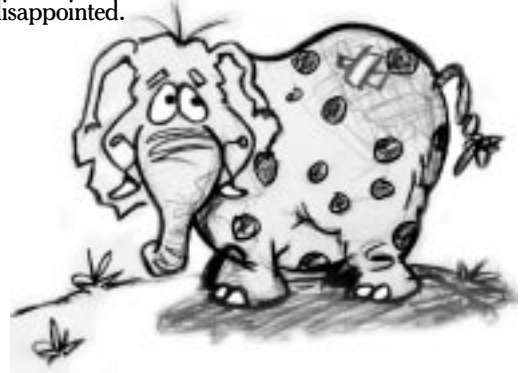
Why This Is Important

Because COTS manufacturers are driven by “first to market” concerns, the COTS product you thought you would get is seldom what actually is delivered. Building your own development around it is always a risky prospect. However, a judicious use of COTS can make a difference in your project’s success. Make your choices wisely, and keep in mind the following:

- COTS may cost you as much over the lifecycle as custom code.
- Integration of COTS is a non-trivial task. The difficulty

goes up exponentially with the number of COTS products. Even the best products cannot be used “out of the box” except, perhaps, in a standalone mode.

- You will not be able to substantially influence or depend upon the evolution of COTS products.
- COTS products evolve asynchronously with one another and with your needs. You will not be able to “freeze” your COTS baseline for very long. Allow ample time in your schedule and budget to accommodate this.
- Expect to live with bugs in COTS products indefinitely. You won’t be disappointed.



Benefits

- May be a cheaper and perfectly acceptable technical solution.
- Frees your developers for custom code.

How Best to Emphasize

- Consider parallel developments and prototypes for COTS products that are not yet available.
- Investigate thoroughly the pricing structure of the product before buying. Site licenses or “concurrent user” limits are usually more cost-effective than seat licenses. Some products are priced attractively but cost a fortune to maintain.
- Select established products with large installed bases to increase the likelihood they will still be there when you deliver.
- Make sure your budget covers the complete cost of integrating and maintaining COTS products.

- “Fly before you buy.” Insist on a trial period for a COTS product and TEST, TEST, TEST to see what it does and does not do. Include labor cost and schedule to cover the trial period.
- Adapt your requirements to COTS capabilities. Wrapping the COTS product in custom code to improve or expand its functionality is extremely risky.



Universal Truth #9

COTS products are not necessarily the best solution. They bring as many risks as benefits. Understand both and make sure your developer does too before adopting.

10 Cutting Edge and Bloody Mistakes

Avoiding Unproven Technologies in Development

Why This Is Important

Cutting-edge technology, by definition, is “out in front.” That means there is a high probability of schedule delays, lack of trained resources to use the new technology/methodology, and few CASE tools that support the product. Interfaces are undefined and may remain open long past the time when they should reasonably (and inexpensively) be closed. Further, this is a most uncertain time for the technology vendor. If the product is not an immediate commercial success, its longevity and promise for the future are at risk.

Few large-scale development projects can react quickly enough to survive the pace of change associated with cutting-edge technologies. Don't be a technology junkie! Think conservatively before you leap!



Benefits

- Avoiding cutting-edge technologies will have significant cost savings over the lifecycle of the system.

How Best to Emphasize

- Use only established technology, methodology, programming languages, and products.
- If necessary, reduce requirements/performance to avoid cutting-edge technology. If that is not possible, select products that espouse adherence to “industry standards” and “open architectures.”
- Make sure you have or add margin (schedule, cost, and performance) to the program (it will cost more, take longer, and still do less than originally intended).
- If you believe you must use them, carry unproven cutting-edge technologies as your No. 1 risk item. Track their development continually and intimately and you will avoid unpleasant surprises.



Universal Truth #10
Cutting-edge products lead to
bleeding-edge projects.

11 Move Over, You're Obsolete

Hardware & Operating System Upgrades

Why This Is Important

Expect to double the processing capacity of the proposed platform by the time you deliver in order to achieve the required performance. In addition, for every 18 months of development time throughout the life of the program, expect to migrate to a newer version of hardware and operating system. Allow time in the schedule and cost in your budget. Because of this periodic migration, defer procurement of the deployed system as long as possible to avoid delivering last year's great idea. Ensure your development approach can accommodate last-minute upgrades.



Benefits

- With sound planning, you might actually deliver a system that's not obsolete.

How Best to Emphasize

- Plan for upgrades in your cost, schedule, and development approach. It's a fact of life.
- Pick a reliable COTS vendor.



Universal Truth #11

Hardware and software system baselines (as well as operating systems) will become obsolete and require upgrading during the development cycle. GET OVER IT!

12 Reheated Spaghetti Is Still Bad Code

Incorporating Re-Use in Your Development

Why This Is Important

Software re-use can reduce costs when re-use is included in the initial planning. Although typically associated with object-oriented technologies, the concept has been successfully demonstrated with other design approaches. Much has been written about the methodology for re-use in an object-oriented design. For other design methods, the probability of successful re-use is inversely proportional to the size and complexity of the re-used module. It is most successful for algorithms based on unchanging physics such as ephemeris work. The intent for re-use must be designed into the original version of the module.



Benefits

- Potential savings in development time.
- Potential savings in cost.

How Best to Emphasize

- Intent to re-use code should be in the initial plan – not a desperate grab at a lifeline.
- Explicitly document the benefit, including the magnitudes and limitations, prior to committing to re-use.
- Manage re-use as a risk item – formally.
- Check other examples of re-use on similar projects.
- Make sure maintenance support is available and responsive enough to support your development needs. Avoid “self-maintenance” by your development team; this will cause divergence from the “golden copy” and you will have custom (not re-used) code.

- Investigate the stability of the current versions.
- Do not commit to create reusable code unless your project has the schedule and budget margin and the commitment of your management.
- Be careful with the amount of special code or “glue-ware” that is required to incorporate re-used code.
- Reward the developer for effective re-use.



Universal Truth #12

Re-use costs extra in the initial development in terms of dollars and schedule as well as to the user in terms of system flexibility.

13 No Free Lunch

Using Automatic Code Generators to Cut Development Time

Why This Is Important

Code generators are beloved by developers, especially those who tend to focus on “lines of code” as a productivity measure. Code generators offer the apparent prospect of “easy” modification; but by their very nature, they produce excessive (read “inefficient”) code, creating significant performance problems that you may not be able to overcome with more powerful hardware. They also contain problems of inflexibility and constraints. Elect to use them only after you understand the risks and have determined that the risks are acceptable.



Benefits

- Potential savings in development time.

How Best to Emphasize

- If, despite your best effort, the developing contractor insists on using code generators for components that are critical to timeline performance requirements:

- Insist these algorithms be prototyped using the code generator, and conduct performance benchmarking tests during the design phase.

- Test for compliance with timeline performance requirements as early as possible.

- Allocate cost and schedule for reworking code to bring the performance into compliance with requirements.

- Require detailed allocation of timeline performance requirements down to the unit level.

- Account for tuning of automatically generated code to enhance performance.

Note: The same applies to code generated by humans (or programmers), but the need is even more critical for code generators.

Universal Truth #13

Avoid using automatic code generators where timeline performance is critical.

14 You'll Have Nobody to Blame but Yourself

The Pitfalls of Government - Furnished Equipment

Why This Is Important

The Government is generally not equipped to conduct large-scale procurements in support of development. Missed deadlines, short suspenses, and logistical nightmares tend to be the norm. Whatever the potential savings may seem to be, the real savings are generally nonexistent. Unfortunately, most GFE delays that do arise hit you late in the game when delivery schedules are critical and program delays are imminent. And it is YOU who are left holding the proverbial BAG. Many a development has been delayed by late or incorrect GFE.



Benefits

- Potential savings in development cost.

How Best to Emphasize

- Negotiate reduced fees on costs used for contractor-furnished procurements.
- Arrange for the contractor to make use of Government purchase agreements wherever there is a price advantage to do so. Often the contractor can get better price breaks than the Government. Include hardware and software procurement in the development contract.
- Understand your responsibility to ensure that GFE is provided on time and that it meets performance requirements.

- Assume GFE will be late and have a different interface than advertised.
- Check with your contracting officer for advice on existing purchase agreements, including the terms of those agreements.
- Establish minimum acceptability criteria for the contractor to procure equipment.

Universal Truth #14

Government - furnished equipment shifts the risk from the developer to the Government.
Leave the risk with the developer.

15 Self-Eating Watermelons

Using Multiple Development Contractors

Why This Is Important

Do not allow your project to become a multi-contractor assembly line. An approach where each contractor adds a part as the software moves down a production line works well for cars, but not for software. Each set is essentially GFE from the last contractor to the next. This is bad because only the Government has any accountability and each contractor is successful whether the overall project works or not.



Benefits

- Concentrating software development within one organization will foster consistent development methodologies, tools, standards, and conventions.
- Employing a single development contractor will make subsequent integration and testing easier.

How Best to Emphasize

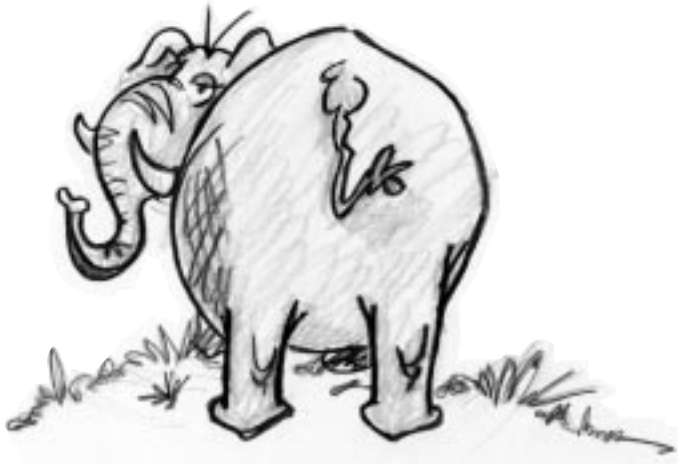
- If splitting software development responsibilities among different organizations is unavoidable, ensure that the interfaces between distinct deliverables are simple and clearly defined:
 - Deliverables should be testable, including interfaces, in a standalone mode prior to delivery for integration.
 - Include cost and schedule to develop and deliver the necessary drivers to conduct such testing.
 - Keep the skill base necessary to maintain and correct all code until final transition.

- Above all, avoid like the plague any situation that has modules being delivered and redelivered back and forth between contractors.
- Ensure someone besides the Government is fully responsible for each discrete program component.
- Subcontracting is almost a given these days - get over it. Make subcontractor management a key clause in the contract and award fee structure. Specify that subs are treated as part of the team and not just as suppliers.
- If you employ multiple or geographically dispersed contractors, identify this in your risk management plan.

Universal Truth #15
Avoid ping pong developments.

Chapter Four

Finishing Touches



16 Nothing Short of Complete Victory

Transitioning Everything to Operations

Why This Is Important

It's certain that if your developing contractor found test tools and test data necessary, operations and maintenance personnel will find them even more so. Operations and maintenance personnel are less familiar with the delivered code. Understanding and planning for this certainly will minimize delays, cost overruns, and unnecessary animosity between developers and maintainers.



Benefits

- Your operations and maintenance organization will have the skills and tools it needs to be successful.

How Best to Emphasize

- Obtain agreement from the O&M organization regarding how relaxed the documentation standards can be for test tools, drivers, and data.
- Include in the RFP the requirement to deliver the test tools, drivers, and test data to the O&M organization.



Universal Truth #17

Deliver to the O&M organization all tools, drivers, and test data used during development.

17 When the Air Tastes Like Water, You Know You're Drowning

When to Go for Help

Why This Is Important

Some programs, as conceived, are simply not executable. For whatever reason – political, budget, hubris, etc. – a well-intended project might become what could be termed a “Death March” project. These programs all have a set of common characteristics such as too much, too fast, too little resources, and/or too little trade space. Such programs break people and reputations. Avoid them at all costs.



Benefits

- Cost savings are recognized when Death March projects are killed early.

How Best to Emphasize

- Honestly assess what it takes for your project to succeed.
- If you have a Death March project in the making, renegotiate the underlying tenets of the project. If this is not possible, get an ironclad guarantee for a future, regardless of the outcome of the project, and a guaranteed medal if you succeed.
- Consider a career change. Maybe in hardware – it's easier.



Universal Truth #16

Know when to hold 'em, when to fold 'em,
and when to RUN away.



References



References

Bate, Roger, et al., November, 1995 "A Systems Engineering Capability Maturity ModelSM," Version 1.1, Maturity Model, SECMM-95-01, CMU/SEI-95-MM-003, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

Curtis, Bill, and William E. Hefley, Sally Miller, September 1995, "People Capability Maturity ModelSM," Maturity Model, CMU/SEI-95-MM-02, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

DeGrace, Peter and Leslie Hulet Stahl, 1990, "Wicked Problems, Righteous Solutions: A Catalogue of Modern Software Engineering Paradigms," Yourden Press Computing Series.

DeMarco, Tom, 1997, "The Deadline," Dorsett House Publishing, New York, NY.

Department of the Air Force Software Technology Support Center, Weapon Systems, Command and Control Systems, Management Information Systems, June 1996. "Guidelines for Successful Acquisition and Management of Software-Intensive Systems: Volume 1 - Version 2.0."

Donaldson, Scott E., and Stanley G. Siegel, 1997, "Cultivating Successful Software Development: A Practitioner's View," Prentice-Hall, Upper Saddle River, NJ.

"Industry Guidelines for Earned Value Management Systems: A Guide for Establishment and Application of an Integrated Management System with Coordination of Work Scope, Schedule, and Cost Objectives and Application of Earned Value Methods for Program or Enterprise Planning and Control," Draft ANSI Standard, June 16, 1997.

Kemps, Robert R., 1992, "Fundamentals of Project Performance Measurement," San Diego Publishing Co., San Diego, CA.

McConnell, Steve, 1998, "Software Project Survival Guide," Microsoft Press, Redmond, WA.

Office of the Under Secretary of Defense for Acquisition and Technology, Joint Logistics Commanders, Joint Group on Systems Engineering, "Practical Software Measurement: A Foundation for Objective Project Management," Version 3.1, April 17, 1998.

Paulk, Mark C., and Charles V. Weber, Suzanne M. Garcia, Mary Beth Chrissis, Marilyn Bush, February 1993, "Key Practices of the Capability Maturity Model SM, Version 1.1," Technical Report, CMU/SEI-93-TR-025, ESC-TR-93-178, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

Putnam, Lawrence H., and Ware Myers, 1997, "Industrial Strength Software: Effective Management Using Measurement," Washington, D.C., IEEE Computer Society Press.

Software Program Managers Network, June 1998, "The Program Manager's Guide to Software Acquisition Best Practices," Version 2.2.

Thayer, Richard H., editor, 1997, "Software Engineering Project Management," 2nd edition, IEEE Computer Society, Los Alamitos, CA.

Yourdon, Edward. "Death March: The Complete Software Developer's Guide to Surviving Mission Impossible Projects.," 1998 Prentice Hall, PTR., Upper Saddle River, NJ.



Elephant Bungee Jumping

A Software Project Manager's Survival Guide



*National Reconnaissance Office
14675 Lee Road • Chantilly, VA 20151*