Conference Paper

Illustrative Example of Flight Software Estimation

Robert A. Georgi

St. Louis
03 June 2009

Our objective is to relate some lessons learned from a recent software cost estimating exercise

- Will focus on using Galorath's SEER for Software (aka SEER-SEM) model
- Will assume audience familiarity with the tool
  - Modeling these issues can be conducted in other tools
- Will be limited to a few key topics in the interest of time

As background in 2008 Booz Allen created the first independent estimate for a new manned spacecraft development effort

- A large aerospace prime contractor is responsible for the development and production of the software using a diverse industry team spread across several states
- Until this time the agency had been relying on an internally developed high level parametric estimate developed before contract award
- The initial Booz Allen estimate was reviewed and refined over several months but the key finding was that significant deltas existed between the project's software development budget and estimated costs especially given requirements uncertainties, schedule slips already experienced and technical challenges
  - We could now quantify the risks associated with the development effort

We found two knowledge bases and two parameters that were areas worth considerable investigation

- Knowledge Base (KBase)
  - Development Standard
  - Acquisition Method
- Parameters
  - Rehost from development to target
  - Retest required
- Regarding the acquisition method we will discuss this extensively and therefore this KBase will be the final topic in the presentation

Within the model KBases provide a quick way to set many parameters to “typical” or “industry norm” values

- Provide a means of setting “reasonable” parameter values when detailed knowledge of the project may not (yet) be available or known
- A given KBase only sets default values for certain parameters
  - Platform KBases set many parameters; most other KBase types set only a few
  - Some parameters are not set by any KBase, and must be manually entered (e.g., Lines of Code entries)
  - KBase settings can be compared/reviewed using SEER-SEM tool

Some parameters will be set by multiple KBase types; in those cases, the last KBase applied takes precedence

- For an example of a KBase applied to a flight software project:
  - Selecting the Manned Space Platform KBase sets the *Test Level* parameter default *Low / Likely / High* values to *Hi / VHi / VHi*
  - Applying the DO-178B Level B Development Standard KBase overrides these values to *Hi- / Hi / Hi+*
  - After selecting/updating KBase choices, review the impacts on parameter values
  - The KBase change may have impacted parameter values you didn't expect it to
- Any given KBase selection can be *completely* overridden by manually setting values of parameters affected by the KBase type
  - User elects to manually change the High value to *VHi-*, yielding the final settings of *Hi- / Hi / VHi-*

Selecting the appropriate KBase for Development Standard relies on an assumption of the required level of certification

- The KBase selection of DO-178B, Software Considerations in Airborne Systems and Equipment Certification is frequently used for space systems

  DO-178B is generally required for commercial airborne systems. The various levels are based on the potential of the software to cause safety related failures identified in the system safety assessment. There are five levels of certification. Level A is the highest.

- In the example of our estimate we initially assumed DO-178B Level A because the prime contractor had indicated it was going to use that certification standard

The examination of our selection for development standard was the first area of contention

- When the project realized how expensive DO-178B Level A certification was going to be, they worked with the contractor to reduce the certification requirements
- Selecting DO-178B Level B, for example, resulted in modeled cost 12% lower compared to Level A

DO-178B Level A

Parameter	Least	Likely	Most
Specification Level – Reliability	Hi+	VHi-	VHi
Test Level	Hi+	VHi-	VHi
Quality Assurance Level	VHi	VHi	VHi

DO-178B Level B

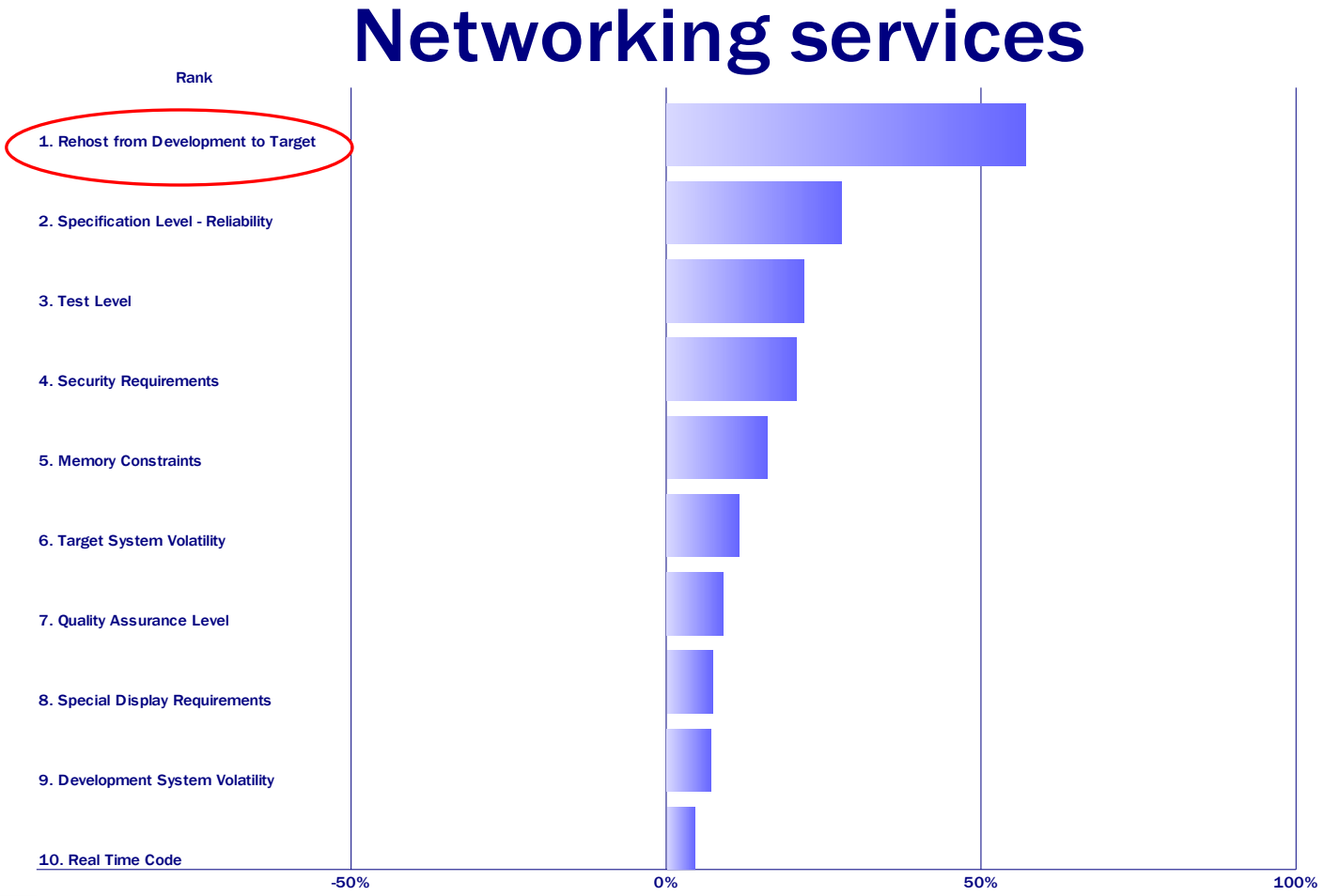
Parameter	Least	Likely	Most
Specification Level – Reliability	Hi-	Hi	Hi+
Test Level	Hi-	Hi	Hi+
Quality Assurance Level	Hi+	VHi-	VHi

If enough data exists consider modifying the KBase parameter settings to best fit your circumstances

- An area being actively explored with both the client community and Galorath is looking at how the development standard KBase affects parameters within the model and review the settings one-by-one for appropriateness
  - Considerations include creating a custom KBase
- It is important to realize that there is rarely a trend for requirements related to safety in government development efforts to decrease over time
- The headquarters of this particular client, for example, is considering draft requirements that would impose new software engineering documents such as a maintenance plan, assurance plan, data dictionary, user manual, and peer review reports
- In the end, Level A may be appropriate for a conservative upper bound, whereas Level B may be a more appropriate given a literal reading of agency requirements

Of the parameters we will examine, the one with most impact is Rehost from Development to Target

ILLUSTRATIVE



The level at which one sets this parameter is a question of the desired fidelity during development

> *Rates the effort to convert the software from the development system (computers, operating systems, etc.) to the target system on which the software will execute. This is related to the difference between the development and target environments, including both hardware and software considerations.*

- Where we to use a platform KBase of manned space, it would set this parameter to *Hi- / VHi / VHi+*
- The KBase of unmanned space sets the parameter to the same level

The feedback we received from the project was the rehost parameter was set too high

- As explained in the previous slide the KBases control multiple parameters
- The rehost parameter is one of the parameters that is set when the selection of manned or unmanned space is made
  - The rehost parameter is not affected by any of the other KBases (e.g. Application Standard)
- While both manned and unmanned KBase set the rehost parameter to the same setting, the manned space KBase is drawn from a fairly old set of data
- With the changes in software development over the past twenty or so years might this KBase have out dated default settings for the rehost parameter?

Let us look at a virtualized software development environment in more detail with regard to the hardware

- Assume software is being developed for a target system that contains a PowerPC processor, an Ethernet controller, RapidIO, a Universal Asynchronous Receiver/Transmitter (UART), and other peripherals
- Each developer of the software team would typically need this target hardware configuration in order to debug and test their software
- When target hardware is in short supply, or not available, this obviously becomes a bottleneck for the software development team
- With a virtualized software development environment the target application code, the real-time operating system, drivers, firmware may all be debugged, tested, and executed using the virtualized target hardware instead of the physical target hardware

The virtualized software development environment also has benefits with regard to the coding effort

- A virtualized software development environment allows one to run the exact same binary that they would run on the physical target
- This means there is no need for:
  - Real Time Operating System (RTOS)/OS API abstraction layers
  - Stubbed out drivers or firmware
  - Multiple build scripts that build the software one way for production environment and another way for a stubbed-out environment

There are certain features that must be provided for the virtualized software development environment to function

- It must have:
  - An instruction set simulator for the microprocessor(s) in the target hardware
  - Behavioral simulation of all devices in the target hardware that the target software interacts with
  - Connections between, and among, simulated targets and the real-world (e.g., networks like Ethernet, MIL-STD-1553, ARINC 429, SpaceWire, FireWire, USB, ATM, and other mechanisms like disk images, memory images, etc.)
  - The ability to use the same tools (e.g., compilers, linkers, debuggers, IDEs, and RTOSs) and processes that the software developer would use with the physical hardware.

There are many questions that need to be answered when making downward adjustments to the Rehost parameter

- How much credit can you take in doing development towards the target avionics system on a workstation?
  - Will all development share the same Integrated Development Environment (IDE)?
  - Will there be a CPU level emulator?
  - Will there be a board level emulator
  - Will there be a target hardware board sitting in the development workstation
  - Or will the development workstations be directly connected to full avionics platform with I/O devices?
- Assuming the answer is the software will be developed on workstations with IDE directly connected to full avionics platform with I/O devices it may be justified to take this setting down to *Nom / Nom / Hi-* from *Hi- / VHi / VHi+*

Source: Input provided by a Galorath consultant

The project we estimated is relying on using workstations resembling hardware-in-loop testing called SoftSIM

- The prime contractor is utilizing a product called Simics® from a company called Virtutech® and referred to in the client parlance as SoftSIM
- The claim is that it "allows software developers to model hardware so accurately that the software cannot detect the difference between running on real production hardware and running on Simics"
- Our information is that this application has successfully been applied for satellite development
- Resetting the rehost parameter to *Nom+ / Hi- / Hi* was considered as the appropriate setting if one assumed that SoftSIM would work flawlessly and greatly reduce the development effort of converting the software to the target hardware environment
  - We did not consider going down as low as *Nom / Nom / Hi-*
- Resetting the rehost parameter to *Nom+ / Hi- / Hi* from default of *Hi- / VHi / VHi+* reduced the estimated cost by 29%

Having established the 'floor" for the Rehost parameter there are considerations for the risks this may entail

- The previous known examples of SoftSIM working were for processors different than the ones that will be used here
  - Does that entail a risk?
- We have never seen a board go from development to implementation without modifications (anomaly resolution), so which version in the hardware will be simulated?
  - The final developed target environment or the notional to be developed target environment?
  - If the modifications to the hardware are significant in parallel development, would this not cause delays as a new version of the hardware is emulated and the code is re-modified and retested?
- Finally variation of the timing is a big issue
  - A 1% variation could mean there is insufficient fidelity to provide the capability of debugging insight
  - For the project in question there has not been an effort made to investigate this issue due to schedule slips and funding issues

The next parameter we will examine is how much retest to assume for pre-existing code

- Bidders frequently assume that a significant amount of code reused because it is a basis of estimate (BOE) that justifies a lower price which can be beneficial for a competitive bid
- We assumed an acquisition method KBase of modification, minor for the reused code
- That set the retest parameter to *1% / 6% / 12%*

We had to modify this setting because we were told that retest objectives would be much higher

- For safety critical Computer Software Components (CSC) at least 50% of reused software would be retested
- Going from a default of *1% / 6% / 12%* to at least *50% / 50% / 50%* increases the estimated cost by 356%
- What kind of effect this has on your estimate depends on how much code reuse there is in your estimate size inputs

The final KBase we will discuss is the Acquisition Method

- As explained earlier since KBases affect multiple parameters, we broke the CSC down into separate components whether it was going to be reused code or new code
- Reused code was modeled as Modification, Minor
- Depending on the circumstances this code was either classified as Pre-exists, not designed for reuse of Pre-exists, designed for reuse
- The prime contractor indicated that new code would be developed using a code generator
- While SEER for Software provides an option of selecting an acquisition method of code generator the set up is not simple

Exploring auto code generation as a topic reveals that it is a complex area fraught with issues

- Claimed autocoding cost efficiencies are key to successful contract competition
- Autocoding typically produces more code for the same function as hand-developed code
- Decreased effort in requirements/design, negligible conventional coding effort
- Potential increase of testing, validation and verification (V&V) effort on backend
- Increased size results in bigger runtime footprint
- Significant variation depending on particular tools, and the autocoding options used
- For Matlab/Simulink, cumulative anecdotal evidence is 50% effort reduction – but quantitative calibration is lacking.
- For SCADE Suite™, even greater effort reduction has been reported (75%)

Source: Autocode Estimation Issues, Michael Lowry, NASA Ames Research Center, 17 December 2008, Constellation Software & Risk Estimation Working Group presentation

There is no consensus on how to model autocode because of these issues

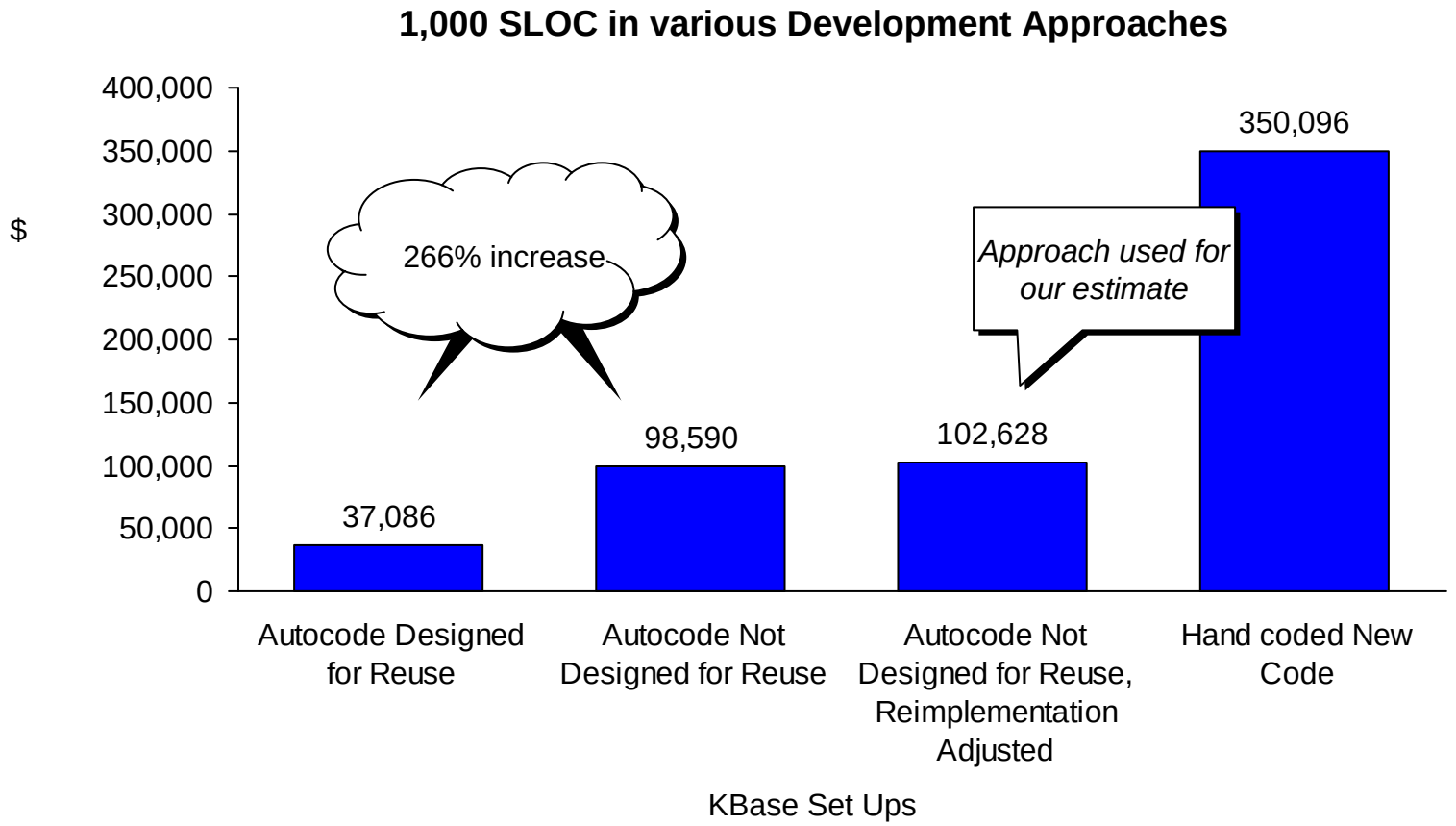
- In the case of our estimate we had the prime contractor's estimate of the resulting Source Line of Code (SLOC)
- Autocode SLOC (aSLOC) are typically some factor greater than hand-coded SLOC (hSLOC) due to inefficiencies in the code generator and its maturity
- Some[1] have modeled aSLOC:hSLOC as a ratio of 2:1
- We chose 4:1 based on our Subject Matter Expertise
- All auto generated code was inputted as *pre-existing, not designed for reuse*
  - Others have modeled autocode as *pre-existing, designed for reuse*
- Using dummy data and setting a reference, the difference between *pre-existing, not designed for reuse* to *pre-existing, designed for reuse* is a decrease of 66%
  - Booz Allen felt *designed for reuse* would be too optimistic

(1) NASA research based on a prototyping exercise, translating legacy X-38 Deorbit Guidance (DG) C FSW (including key math utilities) into Matlab/Simulink artifacts (mSLOC/.m files and Simulink .mdl files)

Compared to developing the same amount of code as hand coded, new code, developing the code as pre-existing code, not designed for reuse resulted in a cost estimate approximately 70% less

- We made one other adjustment to the parameter settings
- KBase of Code Generator default setting for SEER parameter *reimplementation required* is 0.01%
- We over rode this with a setting of 2% for the least, 5% for the likely, and 7% for the most
  - Manual adjustment increased the cost estimate approximately 4%
- The next slide reviews these cost differences in a chart

Comparison of the scale of the implication of the four different approaches referenced in the previous slide



Note: All examples use 1,000 SLOC in a Manned Space Platform for a Flight System Application using a Spiral Development Method and DO-178B Level A Development Standard
Source: Galorath SEER for Software

In conclusion, procurements are won based on the promises of the savings to be accomplished from auto code generation and yet there is significant risk

- Based on the firm's experience in the DoD and National Intelligence environment regarding auto code generation:

- "The consensus was that it requires **more initial** effort to implement using a MDA/MDD tool versus straight coding in a text based language. The overall reduction of effort is primarily based on the ability to develop and assess design approaches quickly prior to implementation and thereby reduce defects upfront. This ability reduces risk that the incorrect design approach was chosen before coding was started. The reduction in resulting 'throw away' code is a reason for a possible overall reduction in effort as is early detection and mitigation of defects that might have been caught in the 'development phase' during unit test, or even in the system test phase."

- We have not observed "quantitatively more productive development, given a relatively high upfront cost to develop the UML artifacts"

- The promise rather seems to be on the reduction in maintenance costs

A software project at the early stage of development introduces more risk than can be handled by the PERT mean

- While SEER for Software does permit entering values by *least / likely / most* ranges, this does not sufficiently instill sufficient range into the estimate to properly define the cumulative distribution function or S-curve for the possible outcome
- The recommended method is to run a set of scenarios to get a range of outputs then use a Monte Carlo simulator like Oracle Crystal Ball, Palisade @Risk or Tecolote ACEIT to get the total range
- For example, rehost, which could drive the estimate down ~30% or up ~50% depending on the starting point, should be modeled by running risk on the estimate of a *Nom / Nom / Nom+* compared to a *Hi- / VHi / VHi+* using a Monte Carlo analysis
- Modeling this as a *least / likely / most* of *Nom / Hi / VHi+* would not permit the internal risk calculator of SEER to properly represent the range of possible outcomes that truly exists

In conclusion, software development has a precedence of adding significant cost and schedule risk to space projects

- According to a recent GAO report the Air Force has only a 50% change of meeting it's launch schedule for the first geosynchronous-orbiting Space Based Infrared System (SBIRS) satellite
  - According to the GAO, the software issue is a risk of $400M per year in cost growth on SBIRS with a total projected cost of $10B
  - The GAO recommended that the contractor adhere to disciplined software practices and that the Defense Department update cost and schedule estimates
- In 1998 the National Reconnaissance Office launched the Future Imagery Architecture project (FIA), a project to build a new generation of spy satellites with a budget of $5B
  - By September 2005, when the project was killed, $4B had been spent without producing a single satellite
  - It was determined that to complete the FIA could cost as much as $18B, $13B more than its original budget
  - Throughout the time period from 1998 to 2002, however, the prime contractor and the NRO continued to insist it could be built within the budget
  - The project was doomed because of lofty technological goals and a tight budget and schedule

I would like to make the following acknowledgements for contributing to the work I presented here today

- The following individuals have provided insight to the estimate and/or reviewed the estimate
  - Gary Constantine, Galorath consultant
  - Gary Hellenga, Booz Allen Hamilton, Colorado Springs, CO
  - Tim Hohmann, Booz Allen Hamilton, Los Angeles, CA
  - Jon Kilgore, Booz Allen Hamilton, Norfolk, VA
  - Denton Tarbet, Galorath consultant
  - Terry Vogt, Booz Allen Hamilton, McLean, VA