Introduction~~n~~

If ~~One of the best ways to determine if your~~you want to see your software ~~is~~ do~~ing~~ what it was designed to do, ~~is to~~ put it in the hands of ~~the people that will be using it~~its users. ~~In the commercial world~~ T~~t~~his is called a beta release in the commercial world. ~~This~~ Beta releases provide~~s~~ the development team~~er~~ with a unique opportunity to see their product perform in the wild and to gain ~~invaluable~~ feedback from the ~~actual~~ domain and functional experts. Its~~The~~ goal is not to have the users redesign the ~~system~~product, but ~~rather~~ to have them validate ~~the product for operational use~~it.

Ideally, prior to entering into ~~any kind of~~ beta ~~program~~, the ~~system~~product ~~has gone~~goes through a rigorous development ~~life~~ cycle and the users ~~are presented~~see ~~with~~ an application that ~~largely~~ meets their needs. However, ~~as one of the immutable laws of the universe~~, no software product ~~is~~ ever delivers~~ed~~ without ~~containing a certain number of~~ flaws. The only ~~real~~ question is~~s are~~ how many and how severe. D~~It is inevitable that d~~efects are ~~will be~~ inevitably overlooked, escape~~ing~~ ~~the~~ testing, ~~program~~ and make~~ing~~ their way to the users. *Your~~The~~ ultimate objective is to minimize this effect and deliver a quality product.*

~~Developing software~~Software development is ~~much~~ different today than it was twenty, ten, or even five years ago. T~~For one thing, t~~he complexity ~~of software applications~~ has risen dramatically. The rise of open source, networking, and computing resources ~~in general, now~~ allow us to do much more than was previously ~~considered~~ feasible. ~~With this~~ I~~i~~ncreased capability provides~~comes~~ increased complexity. ~~This leads to the age old axiom – w~~But with great power comes great responsibility. We need to manage this complexity. ~~Fortunately, there are ways to manage it and this paper will attempt to highlight some of those techniques.~~ It’~~ i~~s no longer sufficient to rely solely on requirements-based testing to gauge ~~the~~ quality and capability ~~of a software product~~. *Quality must be baked into the equation from the start.* ~~This means that both t~~The architecture and the design must become enable~~rs~~ ~~for~~ testing downstream.

Another ~~big~~ advancement in software development ~~has been~~is the homogenization of ~~computing~~ platforms. ~~It used to be a pipedream to imagine w~~”Write~~ing~~ once and run~~ning~~ anywhere” used to be a drea,. Today it’s is not only possible, ~~it is~~but practical. Also, ~~with~~ the ~~improved~~ standardization of interfaces and messaging~~, it has never been~~ makes it easier to encapsulate functionality and design for re-use. Furthermore, as hardware and operating systems become more ~~and more~~ ubiquitous, especially with the maturation of virtual machines, the requirement to validate on a particular platform has become ~~much~~ less of a burden.

Engineering software is one thing, but engineering software under a government contract is another thing altogether. Considering the AEHF Satellite Mission Control System (MCS), ~~this paper~~we’ll ~~will~~ take a retrospective look at ten years of development and reflect on some early decisions that cemented a course wrought with challenges. The purpose is not ~~being~~ to emphasize the challenges, but to highlight the discoveries, improvements, and innovations made along the way. ~~In any go~~Government acquisition of this magnitude ~~there are~~involves ~~numerous~~ many stakeholders ~~involved~~, and ~~from time to time~~sometimes, the wrong decisions ~~get~~ are made for the right reasons. ~~Furthermore, there are~~ C~~c~~ertain pitfalls ~~that~~ can be anticipated but not predicted, ~~;~~ like funding instabilities, anomalies, launch delays, requirements creep, ~~etc~~and so on. These are all part of the greater risk that is faced when procuring a large, complex, "one of a kind," satellite communication system. ~~This risk is again magnified by t~~The massive number of intersegment dependencies (for example, space vehicles, terminals, and ground system) that must ~~eventually~~ integrate and function seamlessly ~~as a whole~~magnify the risk. ~~; i.e. space vehicles, terminals and the ground system.~~

So how ~~do we~~to mitigate ~~this~~ the risk? One answer is to construct a robust acquisition and development strategy that ~~will~~ can withstand the inevitable fluctuations in funding, schedule, and requirements; yet succeed in producing a high quality and highly sustainable system. ~~Unfortunately, h~~Sometimes, hindsight is ~~sometimes~~ the only way to ~~truly~~ understand why ~~these~~ large programs struggle to achieve ~~such~~ their lofty goals. W~~The good news is that w~~e can learn from the past ~~and~~ to make better ~~informed~~ decisions in the future.