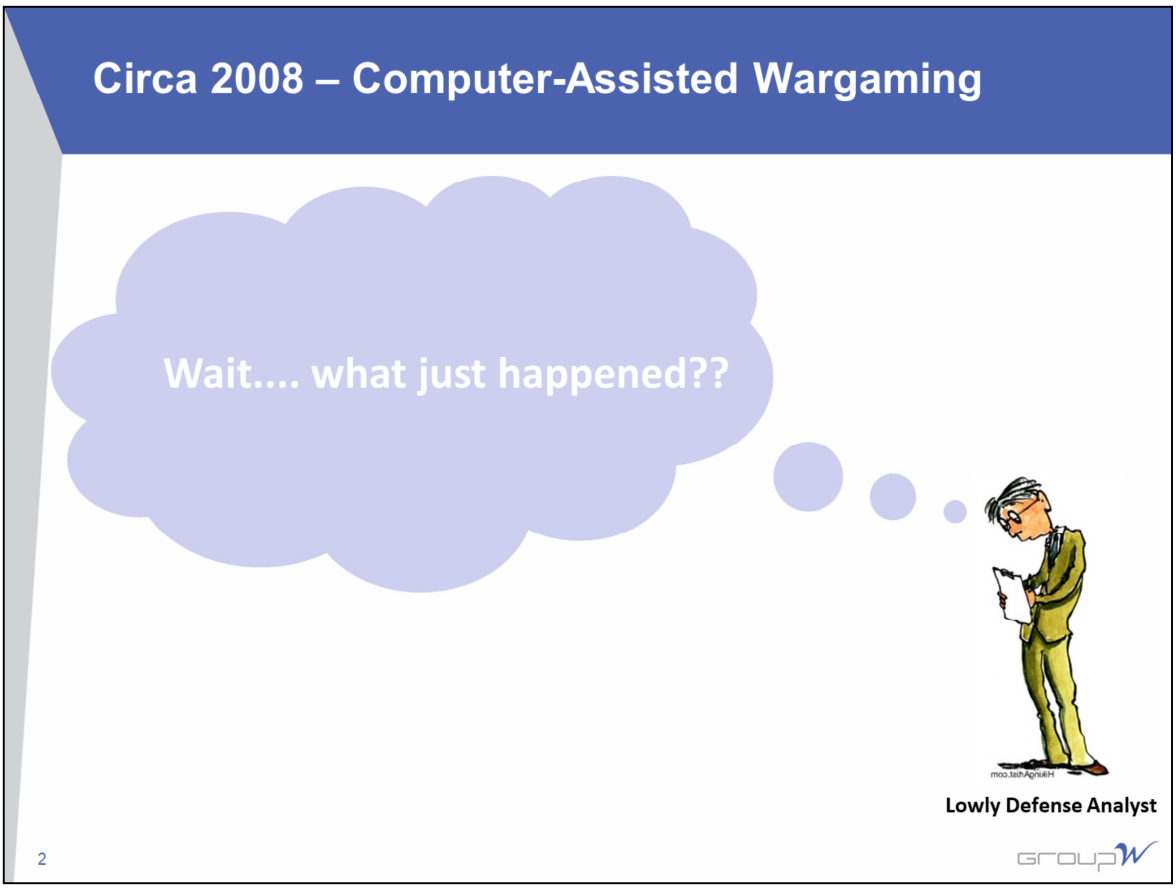
I would first like to express my appreciation for the opportunity to talk to you about some of the work I have been doing over the last ten years integrating technology (primarily software) into US defense wargaming- primarily for OSD Cost Assessment and Program Evaluation (OSD CAPE) and also more recently United States Marine Corps. Today I will be talking about a software engine I designed almost 10 years ago now.

However, I was/am also the lead contractor for the technical development for the wargame repository tasked (beyond the site development) to develop a digital curation approach. I remember back in 2015, I got a call from some of my OSD colleagues. We have to build a wargame repository- how do we start, what should we do- we have hours to get the initial prototype built. I remember suggesting that it sounded like a stunningly bad idea (due to the care and feeding requirements on classified systems)… a few weeks later I was briefing the 4 STAR director of the organization on our progress. Curation remains the most significant challenge and ultimately will determine its long term viability. It is probably worth a case study on the US system.

But what I am going to focus on today is a computer-assisted engine that is used throughout the US Department of Defense and a tool the

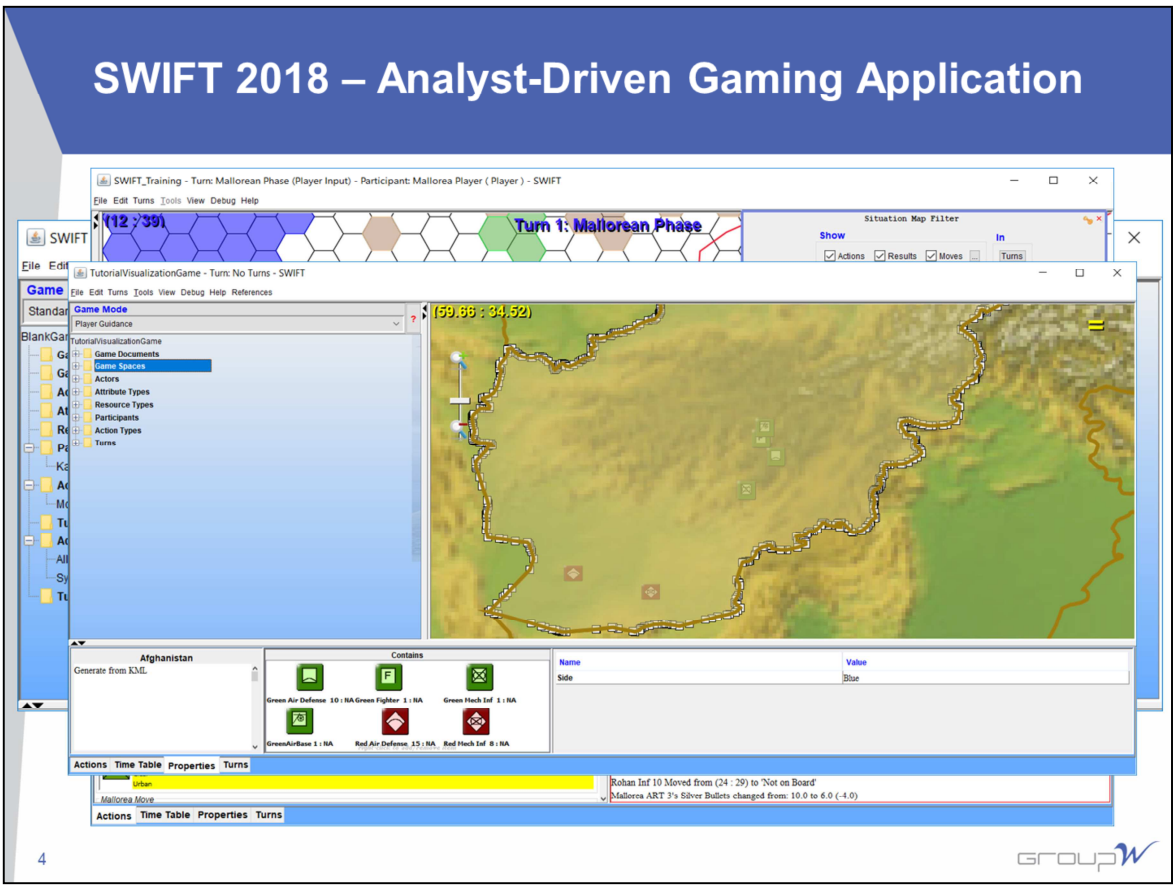
US government has recently shared with the United Kingdom.



In the mid/late 2000s, I started as a contractor within the U.S. Department of Defense in more of an analytical wargaming role. Having read the Art of Wargaming in the late 90s and been a hobby wargamer.... I was startled to, from my vantage point at least, see that wargaming had literally made no significant progress in analytic wargaming. Apart from Microsoft Office products, there were no computer assisted tools available to me to aid my requirement to capture information during the game and certainly no tools to conduct or adjudicate a computer based wargame of my own design. This does not mean that these tools or technology didn't exist somewhere, but the lowly defense analyst wasn't going to see it. The data capture requirement drives you to a computer-assisted platform.

About a year or so later, my colleagues and I successfully made the case for the development of a generic wargame engine to support the design, play, and adjudication of any turn-based defense wargame. I was informed by my experience with play by emails tools such as Aide de Camp, Cyberboard and Vassal, as well as Sid Meir's Civilization II scenario editor, which I considered a brilliant implementation at the time.

The statistical package R and the NetBeans development environment are also important inspirations. NetBeans enabled my ability to program (it made programming accessible and I could author my own solution). R enabled my ability to do statistics and more scientific programming (it made analysis methods far more accessible and I could author my own solution). My colleagues and I proposed an idea on a computer-assisted platform that would minimize the role of the software developer to nice-to-haves or specific game functionality beyond the capabilities of the generic program. We were aiming for a PowerPoint-like capability for wargamers and analysts without any computer programming skills. Design accessibility was a requirement and I thank Dr. Sabin for his outspoken commentary on the matter in the late 2000s. We had design accessibility as a core requirement of the system.



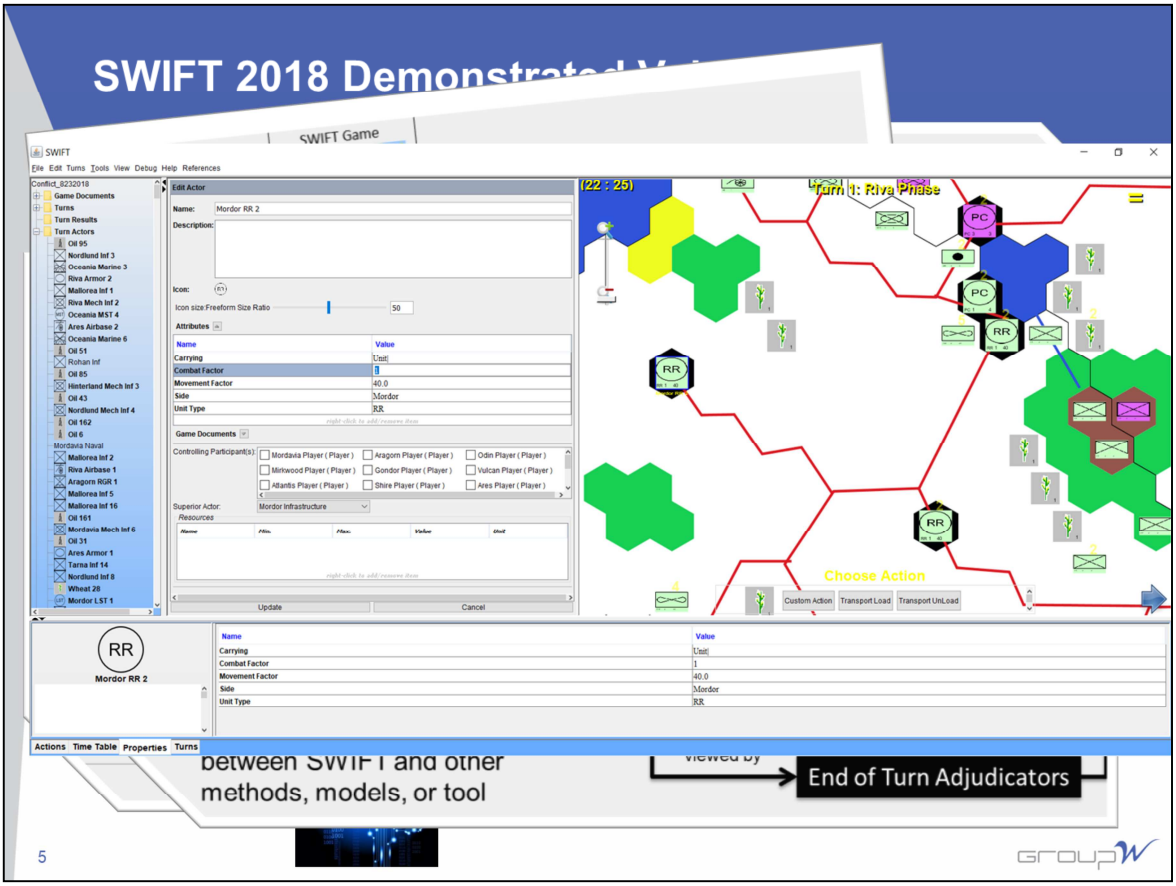
I will return to some of the trial and tribulations of developing a wargaming software engine, but first let me talk a bit about the manifestation of our idea.

SWIFT is a software environment to build, test, play, and analyze ANY turn-based wargame. It was constructed with as much design accessibility as we could imagine at the time. We needed the ability to author our solution, and to change the game rules while the game is being executed. We needed to not commit the user to a particular game mechanic or model, but rather expose to them the ontology of a wargame and let them bring it to life for their purpose.

SWIFT facilitates efficient and consistent game RECORDING. Everything is captured, we had to move beyond the basic ability to author our solution and into the ability to pick it up and read it at any point in time. Ideally, we needed to see not just a God's eye view, but the view the players saw over the course of the game.

Additionally, SWIFT addresses the game VISUALIZATION problem by providing a multidimensional space in which to move pieces. I cannot click on a unit in PowerPoint and immediately see the composition. If SWIFT does nothing else, it must handle multidimensional, temporal, and spatial data.

As a consequence, SWIFT is a multi-sided, hidden information capable, distributed platform with a user experience different when you are the designer, the player, or a game adjudicator. As the designer, I have to see the nuts and bots to bring the game to life. As a player, I need to visualize information and make decisions. As the adjudicator, I have more of a designer view, but I need also need to understand what the different players and adjudicators did during the turn.



This brings me to a consideration of the value SWIFT offers to the wargame design, play, and analysis processes.

Design: When I consider the roll of SWIFT when designing a wargame, three things come to mind. The first and most important to many analysts is its organic data collection ability. As we consider the data collection plan for the wargame, the focus shifts from capturing the game play to capturing the side bar conversations and unstructured table talk. The second is the clarity of thought that is conveyed when interacting with a structured process for incorporating game elements. What do I need players to do and what do they need to know in order to do it are always in the thoughts of a SWIFT game designer. Finally, the most important (from my perspective) is the breadth and depth of mechanics that is available to me to explore the boundaries of realism and playability. The world is at my finger tips; though, as Phil Sabin and others have pointed out, that can be a double edge sword. If you are looking to hang yourself, SWIFT gives you enough rope.

Develop: When it comes to production time whether for play testing or rehearsals, SWIFT's ability to handle some of the more painful physical logistics by allowing for rapid prototyping of digital game artifacts (physical maps, pieces, etc) has been very useful as a time saver. The time spent building event cards, physical counters, and other items during the prototyping stage could be better allocated elsewhere (I never enjoyed arts and crafts). Another area of value is the ability to adjust scenarios given a working rule set. Changing the map and order of battle within a given game's structure is relatively trivial. Ease of repeatability is there for play testing and actual execution is there.

Execute: As I mentioned, players have to assess the environment and create actions (usually dragging and dropping units, and defining actions). Adjudication is handled multiple ways in SWIFT (diversity of opinion was a requirement). The notion was to allow multiple looks at suggested outcomes and allow the "final adjudicator" to decide ground truth. Here is an example of the manual adjudication interface.

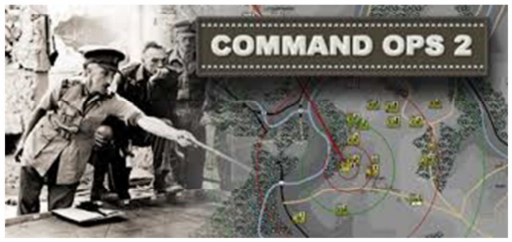
This is an example of how SWIFT execution has looked in some recent wargames. We had three cells (White, Blue and Red) on a local network with White has the host. The white cell would kick off the game and we proceeded through a series of sequential and simultaneous phases within a turn. Hidden information was included so blue and red were seeing their perception of reality (while white would see ground truth). Players were restricted to allowable actions (with the ability to override the rules) and adjudication was semi-rigid with computer-based adjudicators augmenting white cell deliberations. Communication could be tracked within SWIFT in a message board format, but communication was mostly across table with the analysts being actively engaged with the player teams. We had multiple monitors to provide a variety of viewports in the gaming system.

Validate: From a validation perspective, I will give you two examples. Last year I was at a large scale, modified matrix style game, with hundreds of people; we had a supporting role with SWIFT to capture the data and supplement the adjudication. We found some glaring data issues at run time that convinced the sponsor to rerun the event, this time leveraging SWIFT, with a far smaller group to address the issues. In another example, again from last year, we supported a high-level event with many senior leaders that called for a scripted video of wargame results. The data captured in SWIFT from across these games provided an immediate starting point. In either case, the sponsors would not have been able to engage in the richer learning without the automated support offered by SWIFT.

Refine: SWIFT as a computer-assisted platform allows for multiple levels of refinement: (the SWIFT application itself, your specific wargame design, and the specific implementation of your wargame). From my experience, the ability to rapidly adjust visualizations/reports to players as they come up with RFIs that we haven't thought of in advance proved very valuable. Players appreciated the evolution of the player experience as much as the evolution of the model of war.

Share: SWIFT was built for this purpose: to allow for the sharing of methods, game mechanics, and scenarios. There has historically been lots of great work done across commercial and military wargaming, but it often remains elusive in stovepipes of excellence and the wheel is reinvented again. As a community built tool with no black boxes, the instantiated game (as well as executed games) can be shared, scrutinized, and improved.

SWIFT or SLOW



Inevitable comparison with commercial software

The problem is deciding what to automate

Technology is your friend or foe--usually both

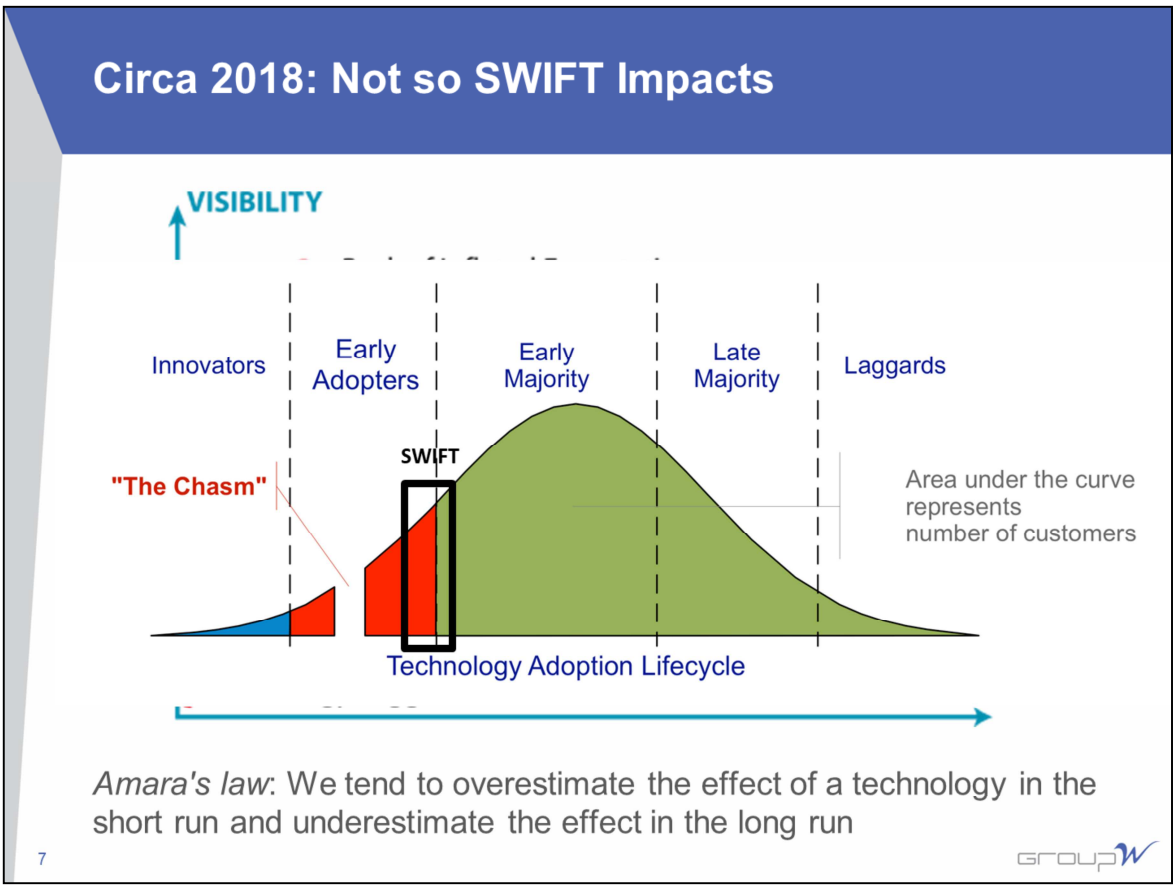
I am perhaps the greatest critic of SWIFT because I happen to use it quite a bit and I know what works well and what doesn't. There are three difficulties that come to mind that are generally true of computer-assisted platforms and SWIFT is no exception.

Comparing SWIFT to Command Ops 2 is far different than comparing SWIFT to Vassal, but it is inevitable. My experience with warfighters is that there is a demand signal for participatory simulation where they issue orders at an echelon and can watch the battle unfold, updating orders as necessary. SWIFT is not a participatory simulation (like Command), though we have incorporated agent-logic at times to manage some lower level behavior. Generally though, you aren't going to see commercial gaming level animations and graphics in a SWIFT game nor are you going to see much agent logic. You may see complicated adjudication, but it built for human players.

When involving a tool like SWIFT, the question is what to automate. With SWIFT you can have a completely manual game setup in minutes-to-hours (the only difference from an analog game is that you are using a digital board); however, there is a tendency to begin to

have the computer take over the execution of player functions and that can add significant cost and time. Shrink-wrapped software release of a computer wargame is not a small endeavor and, more often than not, is not really necessary to support analytic wargaming.

Technology as a friend and foe. You are now constrained to a viewport into the digital system (though I am not advocating against paper products during the game), and the IT infrastructure and the socio-technical design must be thought through well in advance. Play Testing is also very important as the player could potentially kill your game if they manage to do something that you haven't properly handled or considered in advance. Recall the errors from the 1990s (I'm sorry you have done something we haven't expected. The game will now quit). Programmers can extend the engine a great deal, but at your own peril, as I said, SWIFT gives you enough rope to hang yourself. We have a long dynamic list of do's and don'ts… the SWIFT core tends to be very stable, but the analyst built extensions to the game are usually not so robustly



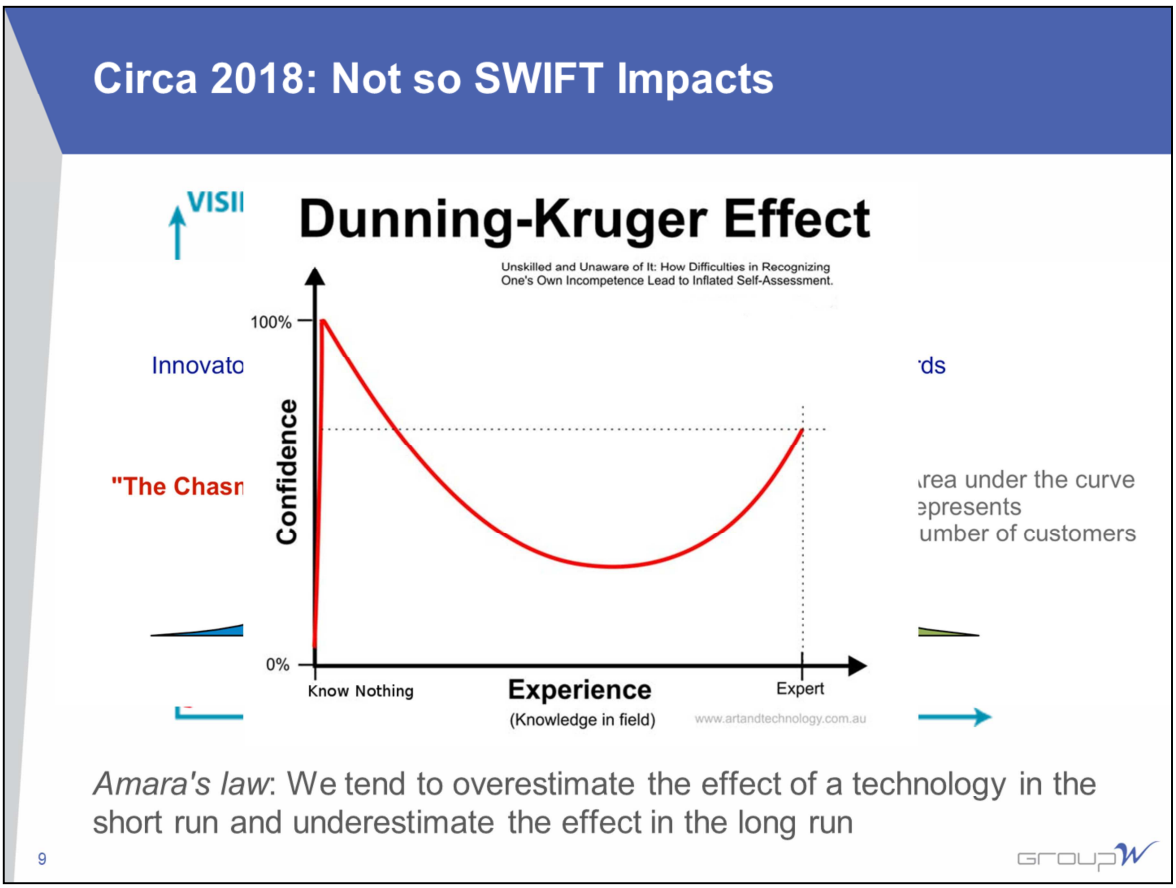
I measure the impact of SWIFT from a technology perspective as opposed its contribution to any particular game. I found the Hype Cycle, which is a branded graphical presentation developed and used by IT research and advisory firm Gartner for representing the maturity, adoption and social application of specific technologies to be an interesting way of discussing it. It mirrors the SWIFT funding profile almost exactly.

We rode the wave of the initial investment in SWIFT, building and selling the plane at the same time… never quite arriving at a product that was worthy of distribution to mainstream users. However, SWIFT was able to weather the valley through a combination of a determined development team, a champion, and the pivot to wargaming that took place within the US Department of Defense. We have been on the up slope for quite some time now and we are not sure where we will peak.

The technology adoption lifecycle is a sociological model that describes the adoption or acceptance of a new product or innovation, according to the demographic and psychological characteristics of

defined adopter groups. This has mirrored our experience with SWIFT development. The chasm is littered with the corpses of forgotten tools. The reason I think we have crossed it (and am here talking about SWIFT) is because the SWIFT idea has withstood several trials by fire relatively unchanged; however the user experience has been significantly enhanced as we have bloodied our nose against multiple organizations and requirements. This experience has been invaluable and I would argue is an absolute requirement for this kind of computer-assisted, game-agnostic platform.

Last slide best slide



I measure the impact of SWIFT from a technology perspective as opposed its contribution to any particular game. I found the **Hype Cycle**, which is a branded graphical presentation developed and used by IT research and advisory firm Gartner for representing the maturity, adoption and social application of specific technologies to be an interesting way of discussing it. It mirrors the SWIFT funding profile almost exactly.

We rode the wave of the initial investment in SWIFT, building and selling the plane at the same time… never quite arriving at a product that was worthy of distribution to mainstream users. However, SWIFT was able to weather the valley through a combination of a determined development team, a champion, and the pivot to wargaming that took place within the US Department of Defense. We have been on the up slope for quite some time now and we are not sure where we will peak.

The **technology adoption lifecycle** is a sociological model that describes the adoption or acceptance of a new product or innovation, according to the demographic and psychological characteristics of defined adopter groups. This has mirrored our experience with SWIFT development. The chasm is littered with the corpses of forgotten tools. The reason I think we have crossed it (and am here talking about SWIFT) is because the SWIFT idea has withstood several trials by fire relatively unchanged; however the user experience has been significantly enhanced as we have bloodied our nose against multiple organizations and requirements. This experience has been invaluable and I would argue is an absolute requirement for this kind of computer-assisted, game-agnostic platform.