

Modeling the Effects of Software-Related Decisions on Early System Cost Estimates: Experience Report from the Software Attributes Trade-off Tool (SWATT) Project

Sarah Sheard
Rita Creel
Patrick Donohoe
Michael J. Gagliardi
Michael D. Konrad
Gabriel A. Moreno

September 2018

SPECIAL REPORT
CMU/SEI-2018-SR-029

Program Name
[Distribution Statement A] Approved for public release and unlimited distribution

<http://www.sei.cmu.edu>



Copyright 2018 Carnegie Mellon University. All Rights Reserved.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute.

This report was prepared for the SEI Administrative Agent AFLCMC/AZS 5 Eglin Street Hanscom AFB, MA 01731-2100

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

Internal use:* Permission to reproduce this material and to prepare derivative works from this material for internal use is granted, provided the copyright and "No Warranty" statements are included with all reproductions and derivative works.

External use:* This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other external and/or commercial use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

* These restrictions do not apply to U.S. government entities.

ATAM® and CERT® are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

DM18-1072

Table of Contents

Acknowledgments	iv
Executive Summary	v
Abstract	vi
1 Introduction	1
2 SWATT Project Results	3
2.1 The Open Architecture Configurable Rating Checklist Tool	3
2.2 The Cost Estimation Tool	3
2.3 Using the SWATT Tool Set	6
3 FY2016 Summary of Activities and Results	7
3.1 Software Cost Estimation tool	7
3.2 Computing estimated costs	8
3.3 Demonstration: Scenarios	9
4 FY2017 Summary of Activities and Results	13
4.1 Security Checklist	13
4.2 Quadcopter Security Issues and Cost of Architectural Improvement	15
4.3 Cost Estimation Results	16
5 Observations and Lessons Learned	19
5.1 Caveat	19
5.2 Observations and lessons learned from the SWATT project	19
6 Future Work	22
Appendix A. Demonstration slide set	23
Appendix B. Software package: Cost estimation using COCOMO	50
Appendix C. Security Architecture (SA) Workbook	55
Appendix D. Initial Security Analysis	58
Appendix E. Quadcopter architecture as revealed by code	62
References	63

List of Figures

Figure 1.	Estimation of Effort using COCOMO II	4
Figure 2.	Using COCOMO II in Estimation Tool	5
Figure 3.	Inputs to Estimation Tool	5
Figure 4.	Output of Estimation Tool	6
Figure 5:	SWATT Approach in FY2016	8
Figure 6:	Cumulative Estimated Yearly Software Costs – Baseline Case	9
Figure 7:	Estimated Costs with Open Architecture Risks Factored In	10
Figure 8:	Estimated Costs of Re-Architected and Relocated FCP Software	11
Figure 9:	Life Cycle Cost Comparison of Two Architecture Trajectories	12
Figure 10:	SWATT Approach in FY2017	13
Figure 11.	Estimation results for 15 years	17

List of Tables

Table 1:	Security Mechanism Categories and Evidence Criteria	14
Table 2.	Cost Estimation Results, first 8 years	16
Table 3.	Cost Estimation Results, last 7 years and total	17
Table 4.	Actual effort implementing change in drone software	18

Acknowledgments

The authors gratefully acknowledge:

- Technical assistance and/or review feedback from Chris Alberts, Felix Bachmann, Stephen Blanchette, Grady Campbell, Brad Clark, Roman Danyliw, Neil Ernst, Peter Feiler, Scott Hissam, Nancy Mead, Tom Merendino, Linda Northrop, Mike Phillips, Fred Schenker, Forrest Shull, Mary Catherine Ward, and Carol Woody.
- Logistical assistance from Heidi Brayer, Julie Cohen, Jill Diorio, Michele Falce, Kevin Fall, Mark Klein, Jim Over, and Jerry Pottmeyer.
- Editorial contributions from Tamara Marshall-Keim.
- Collaboration and contributions from our collaborating organization. Their contributions are highly valued.
- Patience and additional resources for this report from David Zubrow, Charles Holland, and others on the SEI management team.

Executive Summary

Although today's major systems rely heavily on software-enabled capabilities, many defense programs look first at the physical items to be developed, assuming the contractors for those items will provide all needed software for the capability. But software by its nature spans physical items. As a result, software engineering considerations are not addressed in early system trades.

Software is unduly affected by hardware-first early decisions partly because the main way information is communicated today, within and external to a military system, is through software. Software is required to knit together the information about programmatic decisions made by different contractors, which tend to be optimized to their physical components. This may make the software design unable to be efficient and easy to modify; in fact, constraints may prevent the software from meeting appropriate quality goals, as quality attributes are enabled by the software architecture. The resulting design may have significantly worse software complexity, quality, and life-cycle cost than another design could have. Thus, uncertainty, risk, and cost overruns are practically designed-in on such programs.

If software concerns are addressed at the same time as the physical solution is conceptualized, it is much easier to choose a slightly different physical system whose software architecture is tuned to optimize the ability to provide required capabilities.

The Software Attributes Trade-off Tool (SWATT) project was a two-year (FY2016 and FY2017) investigation into the feasibility of incorporating the effects of software architectural decisions into early estimates of system lifecycle costs, using the expertise of a collaborating government organization.

SWATT began in FY2016, with a focus on how the openness of a software architecture affects system costs. The project developed a proof-of-concept implementation and demonstrations for a collaborating government organization. The FY16 SWATT project created a modified version of an open architecture configurable checklist tool and an implementation of the Constructive Cost Model (COCOMO II) estimation equations. The parameters of the cost model are adjusted by the quantified ratings of openness and security that are computed by the checklist tool. The tool chain permits reasoning about the effects of architectural modifications on system costs in several ways: by comparing a baseline system with one modified to support openness or security; by comparing different architectural approaches to achieve the same architectural quality; or by investigating the tradeoffs of openness versus security.

The FY2017 work focused on the domain of drone avionics and extended the cost focus to include some aspects of the architectural quality attribute of security.

Ultimately, the aim of this work is to equip early program staff with as much insight as possible into the implications on the software development effort of early system decisions. Programs sometimes are unaware that early program decisions have specific software architectural implications. A comprehensive SWATT-like tool would clarify these early decisions, and quantify them to the point where their downstream effects on system lifecycle costs could be conclusively demonstrated.

This report also includes a number of lessons learned on the SWATT project.

Abstract

This report describes the results obtained and the experiences gained on a two-year (FY2016-2017) SEI project named the Software Attributes Trade-off Tool (SWATT) project that investigated the relationship of software design decisions to key system-level performance parameters and costs. The FY2016 project created a tool that takes the results of a software openness checklist and converts it into an estimate of software development and maintenance costs. The FY2017 project evaluated how to address the security issues that mission sequences create, when the scenarios to be used depend on architectural features. With significant analyst assistance, this tool can be used first to determine what security concerns a given software architecture may have and then to estimate development and maintenance costs for a number of architectural options that improve security countermeasures.

1 Introduction

In the pre-Milestone-A phase of the system engineering lifecycle, analysts compare potential solutions in trade-off studies. The DoD has invested significant funds to improve modeling and simulation of these solutions, enabling millions of design choices to be evaluated against required capabilities. The goal is to narrow the solution space to the most promising designs long before investing in physical prototypes. However, the design choices modeled today concentrate on physical items and almost entirely exclude software considerations. As a result, decisions are made that intensify software development complexity, increase the difficulty of developing software that meets critical system requirements, and increase system lifecycle cost and risk.

Because software provides an increasingly large and critical amount of capability for DoD systems, its lack of representation in such early decision making is concerning. A reason given for such exclusion, that the contractors who are selected to build the system will also build its component parts including software, implies that software can be handled like other materials. However, our experiences have shown that software's size, extent (connecting everything together) and dynamism (frequent changes) increasingly have a substantial impact on a system's capabilities and cost. For this reason, the Software Analysis Trade-off Tool (SWATT) project was initiated to clarify in what ways early decisions impact software and how this affects system performance and cost.

The SWATT project (FY2016 and FY2017) investigated the feasibility of incorporating the effects of software architectural decisions into a tool chain that models the downstream consequences of those decisions on a system's operation and maintenance (O&M) costs over a period of several years [Sheard 2015, Sheard 2016]. The SWATT project toolkit can be used to analyze software costs (at a rough order of magnitude) for any high-level software architecture, including software architectures that are newly proposed, updates to existing, or legacy.

The SWATT work was performed in collaboration a government organization that is automating the analyses used to select system concepts for potential development. Their computers run a large number of complex programs that take thousands of physical parameters as input, calculate projected system performance, and output system parameters. The results narrow the field from hundreds of thousands of design concepts to a few that can be fleshed out more fully. Currently, all the input parameters used reflect behaviors and contributions of *physical* components: None reflects *software*.

The SWATT project toolkit combines a modified version of an open architecture configurable checklist tool with an implementation of the Constructive Cost Model (COCOMO II) parametric estimation equations [Boehm 2000, USC 2000] to generate cost estimates that take into account the openness and security of the architecture. The checklist tool provides the architectural "scores" that are used to refine the inputs to the cost model. In this way it is possible to estimate the costs of architectural decisions about openness and security against baseline cost estimates generated without the refined inputs. The tool can produce graphical representations of how the estimated costs change over time. The desired outcome is to produce evidence that including software in a system's early trade-off decisions can give a better system architecture that has less risk and less cost (because of avoided rework) over time.

These initial capabilities of the SWATT toolkit were established in FY2016. The existing open architecture checklist tool was an Excel spreadsheet. This was adapted for use on a notional architecture for Army-type fighting vehicles (e.g., a tank). The inputs to the tool were characteristics of the architecture; outputs were ratings on a 0-5 scale of the architecture in terms of modularity, layering, and interface standards. Then these refined parameters were fed into the cost estimation tool, the COCOMO II model implemented in the R programming language. The FY2016 work included demonstrations and a proof of concept for our collaborators [Gagliardi 2016].

FY2017 work focused on a different domain (drone avionics) and extended the FY2016 work to study the architectural quality attribute of security. Analogous to the way in which architectural decisions affect the openness of a system, they also affect many other quality attributes, including security. The SWATT team created a checklist of security mechanisms to address security concerns in the software architecture of a commercial mini drone (quadcopter¹), and incorporated it into the tool to gauge the effects of security-related architectural changes on estimated system costs.

The remainder of this report provides more details on the SWATT project and on the evolution and use of the tools during the two years of the project. Section 2 describes the elements of the SWATT project's tool chain. Section 3 summarizes the project activities and results obtained in FY2016; Section 4 does the same for FY2017. Section 5 provides some observations and lessons learned from the SWATT project experience, and Section 6 briefly discusses possible future work.

¹ A *quadcopter*, also called a quadrotor helicopter or quadrotor, is a multirotor helicopter that is lifted and propelled by four rotors. (Wikipedia)

2 SWATT Project Results

The tools developed by the SWATT project in FY2016 include a slightly-modified open architecture configurable checklist tool (originally created for a government organization) and a cost estimation program that uses the COCOMO parametric estimation equations along with ratings from the checklist tool. The checklist tool is an Excel spreadsheet, and the cost estimation tool is an implementation of the Constructive Cost Model (COCOMO II) in the R programming language as a Jupyter Notebook. The tools developed in FY2017 exist in a draft state; including a comparable security architecture checklist tool. The novelty of the SWATT approach is the linking of architecture ratings to a cost model to produce cost estimates that incorporate software architectural considerations.

2.1 The Open Architecture Configurable Rating Checklist Tool

The Open Architecture Configurable Rating Checklist is an existing tool developed by SEI for a government organization. It is not intended as a self-assessment tool for program offices. Rather, it is intended to be used by an objective observer speaking with project architects to help assess to what extent a system with the proposed software architecture will be able to meet specific program objectives.

The assessment is applied to each “key software component” in a software architecture. The inputs to the assessment are

- A high-level software architecture that identifies key software components, their relationships, and properties. (Software architects provide the necessary explanations and justifications).
- An agreed-upon rating for each criteria (0 to 5, with 5 being the highest) per attribute, per component. (Inspection of artifacts and interviews of system and software artifacts provide the ratings.)

The output is a calculated set of weighted scores (0 to 5, with 5 being the highest) for each key attribute (Modularity, Layering, and Interface Standards) of an assessed software component. The weighted scores (also called ratings) are then used to fine-tune the parameters of the cost estimation tool. (The original checklist tool also included a fourth key attribute, namely Use of Open and Accessible Tools, which this project does not use, as it was deemed less germane.)

2.2 The Cost Estimation Tool

COCOMO II is a well-established, validated cost-estimation model [Boehm 2000, USC 2000]. The cost-estimation part of the SWATT tool chain is an implementation of the COCOMO II equations in a tool (an R program in the Jupyter Notebook application) designed to interface easily with our collaborating organization’s computers (which the scientists program with the R programming language and the Jupyter Notebooks application).

An algorithm was created that uses the ratings from the checklist tool to fine-tune the values of a subset of variables used in the COCOMO II equations, in order to more accurately reflect the effect of open architecture characteristics of a component on the software development effort. The variables chosen for modification were those that would be most affected by architectural characteristics. Our modifications of those variables was limited, in a range set up in the cost estimation program: if the value of the variable was within a range depending on the architecture ratings, we left it alone; if it

was outside the range we changed the value to be the closest number that was in range. No modifications are made to the COCOMO II equations, only to its input parameters, and no changes are made to the range of values for the affected “component” terms (e.g., terms such as SU, UNFM, CPLX, and FLEX).² This interpretation and our modifications of the COCOMO II parameters were approved by an author of COCOMO II, who also provided other useful suggestions.

The R-program produces the software-related cost estimates that can then be combined with hardware costs from the our collaborator’s trade space tool to give a more complete picture of estimated costs than a hardware-only analysis would produce.

Figures 1-4 show our adaptation of the COCOMO II for estimation.

Widget Implements COCOMO II* to Estimate Effort

The cost estimation model COCOMO II requires rating ~30 factors:

- **Software size:** effort spent in writing or modifying code
 - Measured or converted into thousands of lines of source code (KSLOC)
- **Scaling factors:** grow non-linearly with size (economies of scale)
 - Software size is raised to this power
 - Assumed to be project-wide
- **Cost drivers:** increase effort linearly (“effort multipliers”)
 - Assumed to be component-specific (except for SCED)

$$\text{Total effort} = A * (\text{Software_size} ^ [B + \text{Scale_factors}]) * (\text{Cost_drivers})$$

- A and B are constants set by standard or calibration
- Total cost and schedule duration are functions of Total effort.

*COCOMO II Model Definition Manual V2.1 (2000)

Figure 1. Estimation of Effort using COCOMO II

² The SWATT overview presentation provides more details on how checklist ratings were mapped to terms in the COCOMO II equations [Gagliardi 2016].

Using COCOMO II in the Widget

- [1] $AAF = (0.4 * \text{DM}_i) + (0.3 * \text{CM}_i) + (0.3 * \text{IM}_i)$
- [2] $(AAF \leq 50) \text{ AAM} = (\text{AA}_i + (AAF * (1 + (0.02 * \text{SU}_i * \text{UNFM}_i)))) / 100$
 $(AAF > 50) \text{ AAM} = (\text{AA}_i + (AAF + (\text{SU}_i * \text{UNFM}_i))) / 100$
- [3] $\text{SIZE}_i = \text{KSLOC}_i * \text{GRWTH}_i * (1 - (\text{AT}_i / 100)) * \text{AAM} * (1 + (\text{REVL}_i / 100))$
- [4] $\text{SIZE}_{\text{Aggregate}} = \sum_{i=1}^n \text{SIZE}_i$
- [5] $E = B + (0.01 * (\text{PREC} + \text{RESL} + \text{TEAM} + \text{PMAT} + \text{FLEX}))$
- [6] $\text{PM}_{\text{Basic}} = A * (\text{SIZE}_{\text{Aggregate}})^E * \text{SCED}$
- [7] $\text{PM}_{\text{Basic}(i)} = \text{PM}_{\text{Basic}} * (\text{SIZE}_i / \text{SIZE}_{\text{Aggregate}})$
- [8] $\text{PM}_i = \text{PM}_{\text{Basic}(i)} * \text{DATA} * \text{RUSE} * \text{DOCU} * \text{TIME} * \text{STOR} * \text{PVOL} * \text{ACAP} * \text{PCAP} * \text{PCON} * \text{APEX} * \text{PLEX} * \text{LTEX} * \text{TOOL} * \text{SITE} * \text{SCED} * \text{RELY} * \text{CPLX}_i$
- [9] $\text{PM}_{\text{Aggregate}} = \sum_{i=1}^n \text{PM}_i$
- [10] $\text{Cost} = (\text{PM}_{\text{Aggregate}} * \text{LABOR}) / 1000$ (\$K per year, not in COCOMO II Model)

Green = Component based

Blue = Project based

Figure 2. Using COCOMO II in Estimation Tool

Inputs

COCOMO II Project Parameters (Excel .csv file)

- Filename: "Tank_As_Is_NO_OA_Fixed_Input.csv"

OA1	OA2	OA3	OA4	FLEX	PREC	RESL	TEAM	PMAT	DATA	RUSE	DOCU	TIME	STOR	PVOL	ACAP	PCAP	PCON	APEX	PLEX	LTEX	TOOL	SITE	SCED	RELY	CPLX	A	B	LABOR
-1	-1	-1	-1	5.07	2.40	1.41	1.1	3.12	1.28	1	1.23	1.29	1.17	1.3	0.71	0.76	0.81	0.81	0.85	0.84	0.78	0.89	1	1	1	2.94	0.91	25000

COCOMO II Component Parameters (Excel .csv file)

- Filename: "Tank_As_Is_NO_OA_SW_Input.csv"

Name	KSLOC	OA1	OA2	OA3	OA4	AA	DM	CM	IM	SU	UNFM	REVL	PREC	RELY	CPLX	GRWTH	AT
FCP-SM	150	-1	-1	-1	-1	0	100	100	100	35	0.5	0	0	1	1	0.1	0
FCP-FC	250	-1	-1	-1	-1	0	100	100	100	35	0.5	0	0	1	1	0.1	0
FCP-FS	100	-1	-1	-1	-1	0	100	100	100	35	0.5	0	0	1	1	0.1	0
FCP-MMII	150	-1	-1	-1	-1	0	100	100	100	35	0.5	0	0	1	1	0.1	0
FCP-C&C	150	-1	-1	-1	-1	0	100	100	100	35	0.5	0	0	1	1	0.1	0
RT_EXEC	75	-1	-1	-1	-1	0	100	100	100	35	0.5	0	0	1	1	0.1	0

Figure 3. Inputs to Estimation Tool

Output₂ Cumulative Yearly SW Costs per Project

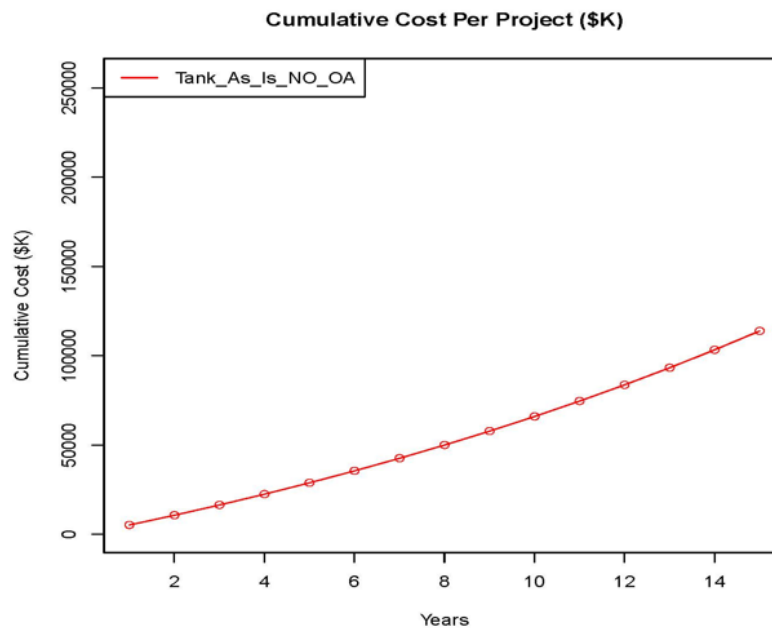


Figure 4. Output of Estimation Tool

2.3 Using the SWATT Tool Set

The SWATT Toolkit demonstration slide package is included here as Appendix A, and the software set used for the demonstration is included as Appendix B.

The original intent was to use SWATT's products to estimate the costs of software architectural decisions for actual early-phase DoD programs. However, disclosure and distribution restrictions with respect to architecture and data caused the project to resort to using legacy data and a notional software architecture for Army fighting vehicles in FY2017, and open-source quadcopter software architecture in lieu of a military unmanned aerial vehicle (UAV) in FY2017. Despite these limitations, the proof of concept demonstrated in FY2016 was sufficiently convincing to our collaborators that they are adapting it for their computers and plan to use it in future estimation efforts.

The next two sections provide more details on SWATT tool chain creation and use in FY2016 and FY2017.

3 FY2016 Summary of Activities and Results

Work in FY2016 focused on the relationship between openness (“open architecture”) characteristics of a high-level software architecture and that software’s contribution to system development and maintenance costs. The open architecture checklist tool was used to assess three open architecture parameters of a software architecture, on a per-major-software-component (CSCI) basis: modularity, layering, and the use of interface standards. Ratings for these parameters were elicited through artifact inspection and through interviews with program system and software architects.

The ratings of the open architecture parameters were used to refine the inputs to the COCOMO II software estimation equations³ in the tool that SEI built in R code. The tool estimates the effort needed to complete the software development and maintenance tasks for a DoD program. With input estimates of labor rates, the tool converts effort estimates into cost estimates, which can then be combined with hardware costs from the our collaborator’s trade space tool to give a more complete, software-inclusive, picture of likely downstream program costs based on early decisions.

3.1 Software Cost Estimation tool

An algorithm was created that maps a component’s set of open-architecture-related ratings from the checklist tool to adjustments in the allowed values of selected variables used in COCOMO II for effort estimation.

COCOMO II^[1] estimates costs by asking questions about different aspects of the software project (such as software understanding, SU), getting answers such as “Very Low” or “Nominal” for each variable (both “Effort Multipliers” and “Scale Factors” for development, and “Effort Multipliers” and “Scale Factors” for maintenance). COCOMO II calls such answers “ratings” for each driver, but in this document the word “rating” is reserved for the output from architectural checklists, so we are calling them “answers”. When computing the estimate, COCOMO II translates such answers into numerical values.

In the SWATT estimation software, we modify the allowed COCOMO II answers for specific variables to account for high (or low) architectural openness. We do not change the effort estimation equation nor what numerical values are assigned to each answer: we only modify which answers may be used to rate the COCOMO variables (I.e. if modularity is high, SU is presumed to be high, so all estimates of SU below High are re-answered as “High”). This interpretation and our modifications of the COCOMO II variable ratings were reviewed at a meeting with an author of COCOMO II, who considered the modifications to be basically sound.

³ One of the authors of the COCOMO II model validated the mapping from the architecture ratings to the COCOMO II parameters.

^[1] Gagliardi’s SWATT overview presentation provides more details on how checklist ratings were mapped to terms in the COCOMO II equations [Gagliardi 2016].

The COCOMO II variables whose ratings were selected for adjustment were those that would be most affected by open architectural characteristics. The variables whose ratings were selected for adjustment to reflect the open characteristics of a component were SU, programmer’s unfamiliarity with the software (UNFM), product complexity (CPLX), and development flexibility (FLEX).^[1]

Trajectories. The estimated software cost per year can be compared for two alternative “trajectories:” 1) keeping the legacy architecture, with its high maintenance cost, and 2) continuing to use the legacy architecture for the time being, while developing a new, more open architecture that will have lower maintenance costs. The first trajectory involves no development cost and high maintenance cost based on the original architecture’s lack of openness. The second trajectory adds the cost of rearchitecting the software as high development cost for the first few years to the continued high maintenance cost for the old architecture during this time. Once the new architecture is in place, the second trajectory shows only the lower maintenance cost for the new, open architecture software.

3.2 Computing estimated costs

Figure 5 shows the intended way to use the envisioned SWATT tool chain to compute these estimated costs. We defined as our system a notional (military) tank with a legacy (non-open) software architecture. For a baseline, one first obtains a naïve estimate of maintenance costs for the current architecture, using the SWATT FY16 cost estimation tool without modifications due to openness. (This tool is called a “widget” in the figure, though we ultimately moved away from that term as we determined that our use of the term was inconsistent with the client’s). Then one calculates a more knowledgeable maintenance cost for the first trajectory by evaluating the openness of the architecture and inputting the checklist ratings into the SWATT FY16 cost estimation tool. For the second trajectory, one considers how to rearchitect the legacy software to a more open version. The expected cost of rearchitecting becomes the development cost. This rearchitected software is re-evaluated for openness using the checklist tool, and then the maintenance cost is re-estimated using the SWATT FY16 estimation tool with input openness ratings.

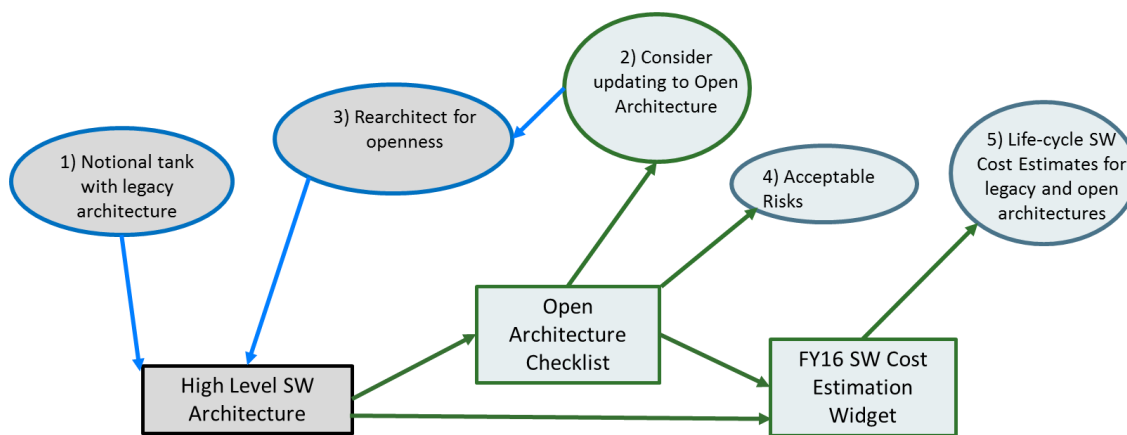


Figure 5: SWATT Approach in FY2016

^[1] Gagliardi’s SWATT overview presentation provides more details on how checklist ratings were mapped to terms in the COCOMO II equations [Gagliardi 2106].

3.3 Demonstration: Scenarios

This approach was the basis for demonstration of SWATT conducted for our collaborators, using the software architecture of the fire control processor (FCP) of a notional fighting vehicle, in this case a tank. Each demonstration postulated a cost-estimation scenario and then ran the tool to predict the cumulative costs over a fifteen-year period [Gagliardi 2016]. The Demonstration slide set is provided as Appendix A. The software that performs the calculations below and in Appendix A is provided as Appendix B.

Scenario 1: A program office acknowledges that the existing FCP software O&M costs are onerous and wants to estimate those costs over the next fifteen years. In this baseline scenario, the cost estimation tool is executed naïvely: without any modification based on openness as rated by the architecture checklist tool. Figure 6 shows the cumulative estimated yearly software costs.

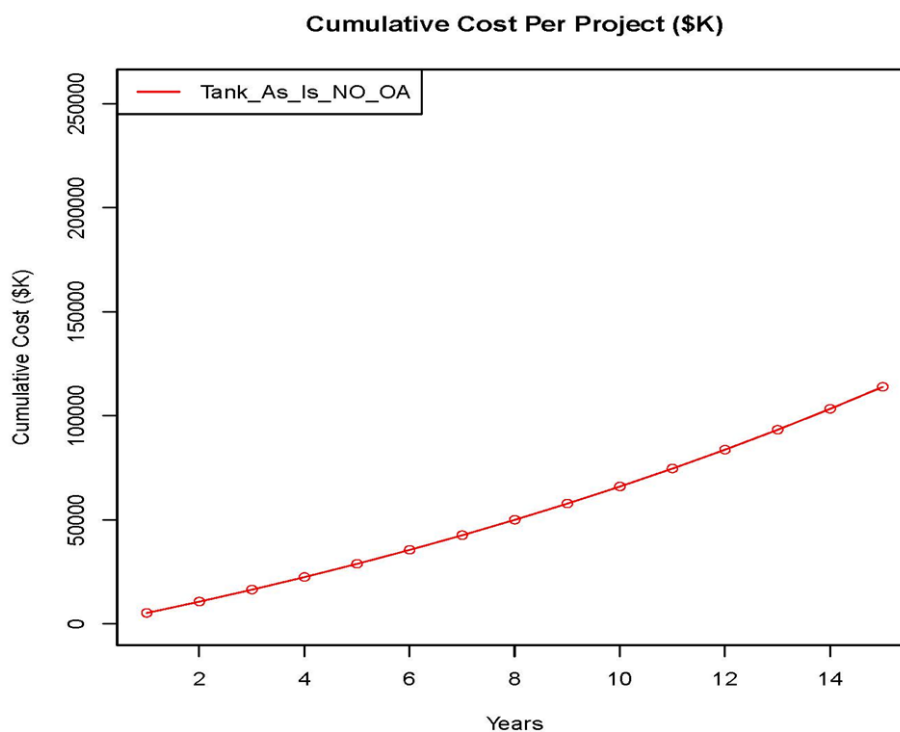


Figure 6: Cumulative Estimated Yearly Software Costs – Baseline Case

Thus the maintenance costs, in the absence of any architectural considerations to refine the estimation process, are estimated at approximately \$110M over 15 years.

Scenario 2: In this scenario, open architecture considerations are factored into the cost estimation. A program office acknowledges that the existing FCP software O&M costs are higher than estimated and are unsustainable, and that they do not currently have visibility into the factors associated with the construction of the FCP software that contribute significantly to the O&M costs. They decide to baseline the existing FCP software architecture risks with respect to open architecture, and account for the open architecture risks in the cost estimations over the next fifteen years. Figure 7 shows the estimated cost graph with open architecture risks factored in.

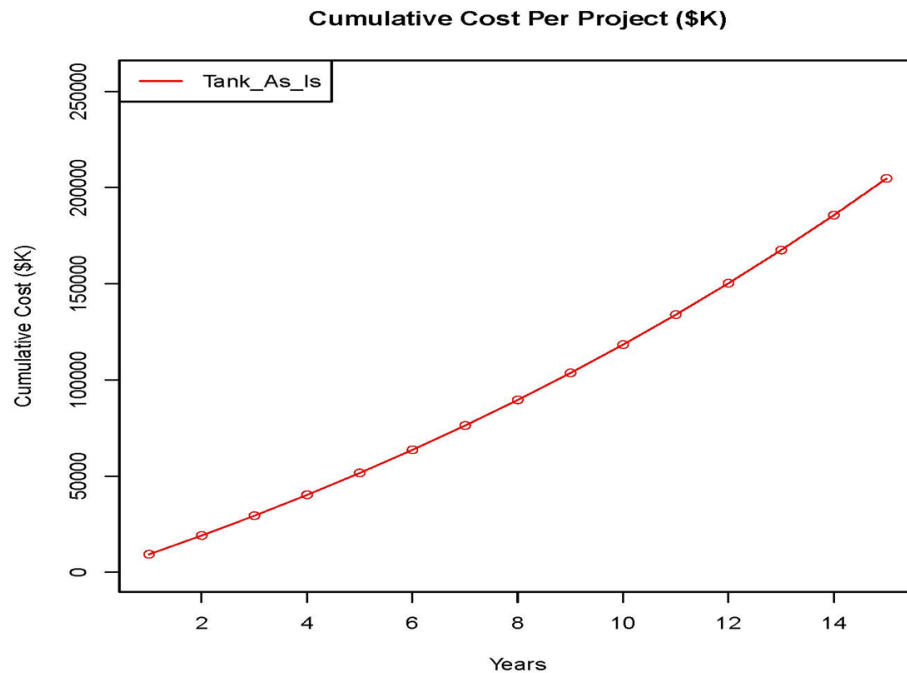


Figure 7: Estimated Costs with Open Architecture Risks Factored In

Now the estimated costs are approximately \$200M after fifteen years. Comparing this with Figure 6, there is an additional \$90M, approximately, in O&M costs after fifteen years that result from factoring in the (not-so-open) open architecture ratings.

Scenario 3: The program office acknowledges that the existing FCP software O&M costs are unsustainable, and now decides that alternative architecture approaches need to be investigated. The alternative is to re-architect the FCP software to take advantage of expected open architecture benefits. To do this they decide to relocate the FCP software out of the turret and into the main processing unit of the vehicle. We redrew the architecture and estimated the size of the new software modules needed to make this happen. We then input the size and other parameters back into the COCOMO II equation.

Figure 8 shows the estimated development costs (red upper curve) and maintenance costs (blue lower curve) associated with this proposed architecture. (Note the expanded scaling of the vertical axes when comparing this with previous figures.)

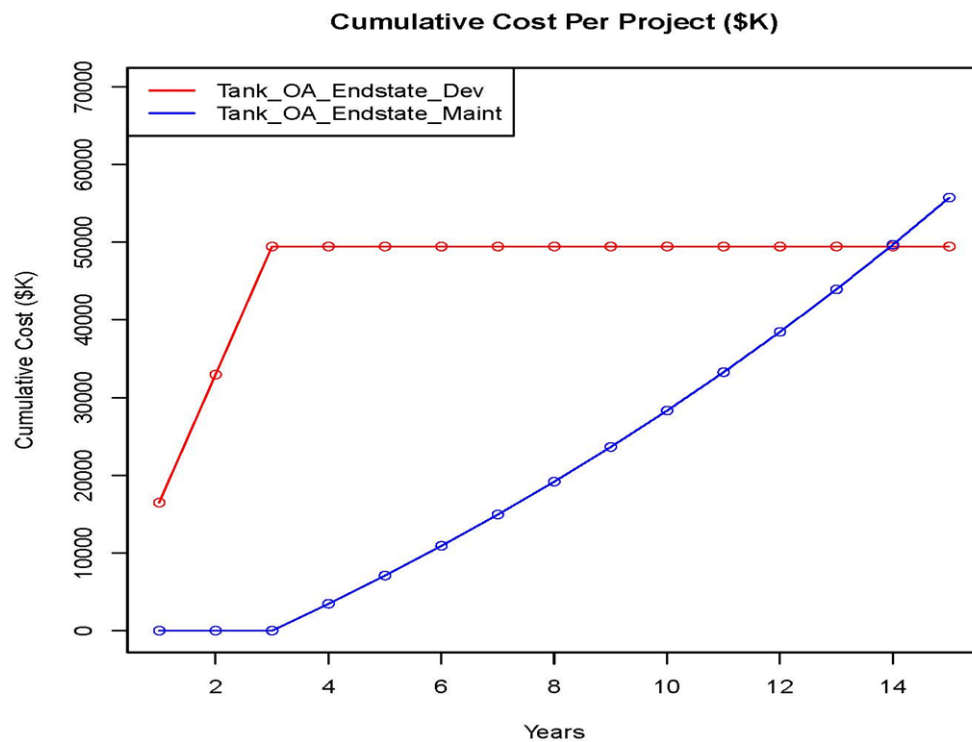


Figure 8: Estimated Costs of Re-Architected and Relocated FCP Software

This figure shows only the cost of developing and then maintaining the new, more open architecture. Development costs (top line) are high for the first 3 years, after which the development cost goes to zero and the maintenance cost is estimated for this more open architecture (bottom line).

Scenario 4: This final scenario compares two architecture “trajectories.”

Trajectory T1

- Continue with maintenance of the existing FCP software for fifteen years

Trajectory T2:

- Continue with maintenance of the existing FCP software for three years.
- Re-architect for OA and implement the re-architected FCP software for three years.
- Cut over to the re-architected FCP software starting in year 4 and maintain it for twelve years.

A trajectories script file is an input to the software cost estimation tool. The trajectories script file contains a number of user-specified projects, with start and end years. The tool generates cumulative, yearly software estimates for each trajectory and plots the data for comparison purposes.

Figure 9 shows the estimated costs of these two trajectories.

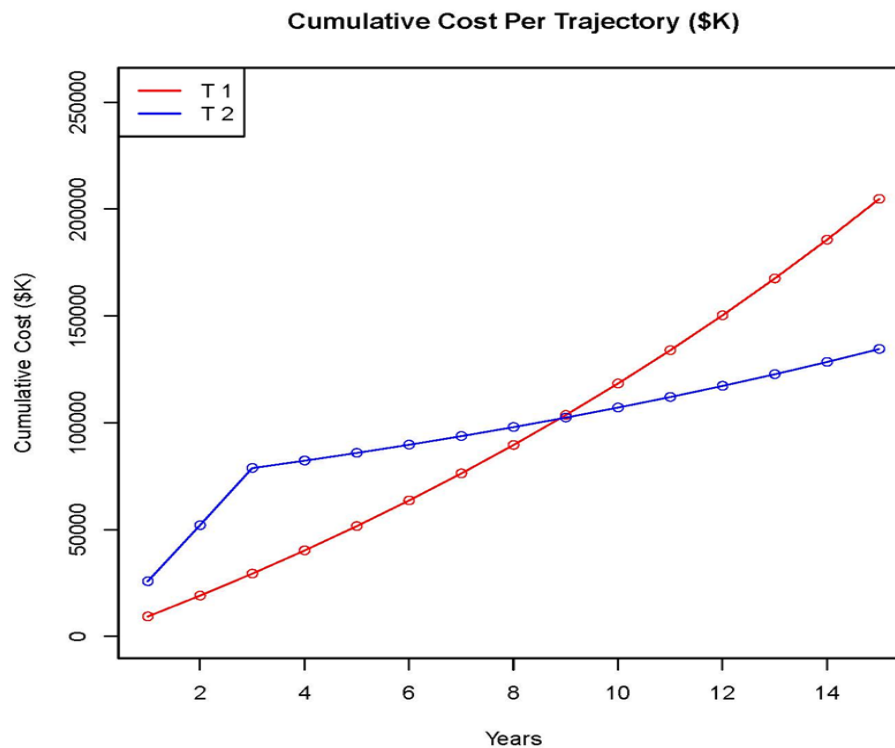


Figure 9: Life Cycle Cost Comparison of Two Architecture Trajectories

For the first eight years, trajectory 1 (red lower curve, continue to use old architecture) is less expensive than trajectory 2 (blue upper curve, build new software and then use it). However, the build-new-architecture trajectory begins to outperform the use-old-architecture trajectory 1 starting in year 9, resulting in approximately \$75M in savings by year 15.

These four scenarios show some of the possible uses of the SWATT tool chain, and the relative ease with which the openness of an architecture can be factored into system cost estimations. Our collaborating organization plans to use the SWATT tool chain in future cost estimation analyses.

The FY2016 work benefited from being able to use an existing checklist tool for architecture openness. The FY2017 work examined an architectural quality—security—for which there was no pre-existing checklist tool. That work is the subject of the next section.

4 FY2017 Summary of Activities and Results

Work in FY2017 focused on extending SWATT to identify the effect of a different architectural quality, security, on early system cost estimates.

Figure 10 is a variant of Figure 5, showing the envisioned way to use the SWATT tool chain in a different domain (drone avionics) to account for different architectural considerations (in this case security) in cost estimation.

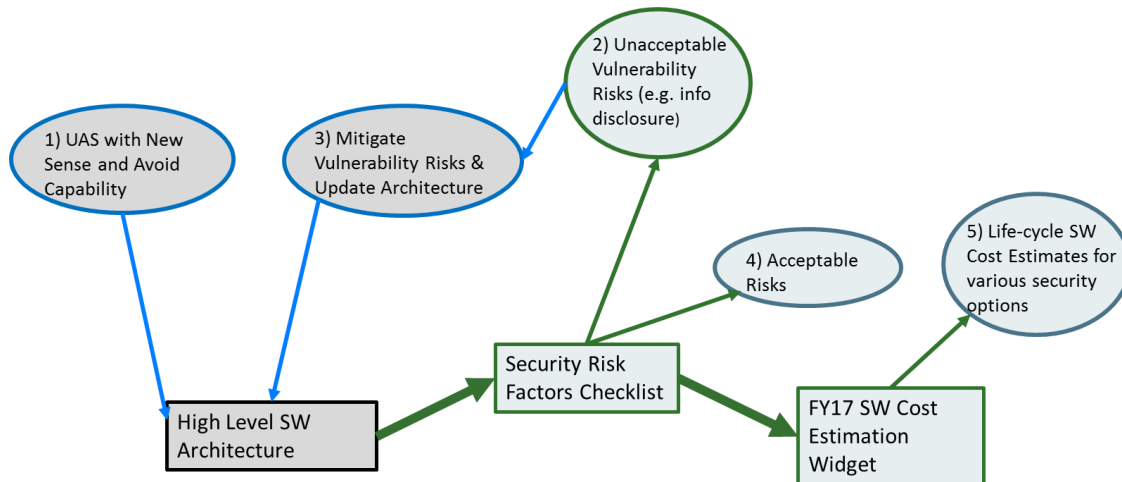


Figure 10: SWATT Approach in FY2017

Whereas the FY2016 work made use of an existing open architecture checklist tool, no such tool existed for security, so one of the major tasks of the FY2017 work was the creation of a draft security checklist tool (see Appendix C for a description). The items in the security checklist differ in structure and derivation from those in the open architecture checklist. Open architecture characteristics are constructive quality attributes (meaning they are determined at design time), whereas security is an operational quality attribute that, despite a fixed design that considers security, can vary depending on the operational situation. The checklist items for open architecture evaluation focus on characteristic *structural attributes* of openness. For security, the checklist items focus on *functional mechanisms* that can be used to implement basic security controls. To develop items for the security checklist, one needs some awareness of the type of system (e.g., IT or cyber-physical), the mission or capability of interest, and the operational context.

4.1 Security Checklist

For the SWATT research effort, the original intent was to create the security checklist by analyzing security threats to an actual UAV (e.g., an unauthorized user attempts to gain control of the drone by exploiting a vulnerability) and interviewing UAV architects. However, as noted earlier, disclosure restrictions resulted in the effort using a commercially available quadcopter instead. The quadcopter

was based on a DJI Flame Wheel F450 frame⁴ with a Pixhawk®⁵ flight management unit⁶ running the PX4 open-source autopilot software⁷. Therefore, the checklist developed for this work is at an early, conceptual stage of development. Appendix C provides an introduction and description of the Security Architecture workbook, which includes a security checklist (in the form of architectural mechanism data) and supporting worksheets that characterize security benefit data and enable rating the resultant checklist selections in terms of the security benefits they confer.

The security checklist comprises a set of six security mechanism categories, each of which has an accompanying set of architectural security mechanisms that might be incorporated in the quadcopter’s architecture. The security mechanism categories were selected from 20 security control families identified by the National Institute of Standards and Technology (NIST); the mechanisms are derived from security controls [NIST, 2014b, 2017b; CNSS 2013, 2014]. The selected controls are termed “mechanisms” to emphasize the fact that they would be implemented in the quadcopter architecture rather than via security procedures.

Security Mechanism Categories. The six security mechanism categories are

1. Access control
2. Audit and accountability (i.e., to enable analysis and detection of events and access attempts)
3. Contingency planning (for safe-mode implementation only; other controls in this category are primarily procedural rather than architectural)
4. Identification and authentication
5. System and communications protection
6. System and information integrity

This set of security mechanism categories is not meant to be exhaustive. Rather, it represents the initial categories of interest applied when evaluating the quadcopter architecture.

Table 1 shows three of the six mechanism categories and one example mechanism for each.

Table 1: Security Mechanism Categories and Evidence Criteria

Security Mechanism Categories	Architectural Security Mechanisms (Controls)
Access Control	The architecture incorporates mechanisms to control access to information and system resources from ground control stations.
Identification and Authentication	The architecture (communication protocol and components) supports authentication of users.

⁴ <https://www.dji.com/flame-wheel-arf>. See [DJI 2016].

⁵ Pixhawk is a trademark of Lorenz Meier.

⁶ <https://pixhawk.org/> See [Meier 2016].

⁷ <http://px4.io/>. See [Dronecode 2016].

System and Communications Protection	The architecture supports capacity, bandwidth, and throughput monitoring and management to limit the effects of information of information flooding denial-of-service attacks.
--------------------------------------	--

Mission Threads. To create the checklist, the SWATT team created a “mission thread” [Gagliardi 2013] for the quadcopter. A mission thread is a sequence of end-to-end activities and events that takes place to accomplish the execution of one or more of a system’s (or system of system’s) capabilities. The team then assessed potential threats and vulnerabilities in each step, and created a set of security threat scenarios for selected mission thread steps. The goal was to identify architecture-level mitigations for security attacks on the quadcopter software. Attack scenarios were identified using the STRIDE⁸ threat model [Microsoft 2005, Howard 2006] supplemented by a literature search on a variety of research papers, NIST work on critical infrastructure cybersecurity and security and privacy controls, and the Committee for National Security Systems (CNSS) standards on controls for national security systems [NIST 2014a, 2014b, 2017a, 2017b; CNSS 2013, 2014].⁹

The initial STRIDE analysis, including mission steps and security scenarios, is provided as Appendix D.

4.2 Quadcopter Security Issues and Cost of Architectural Improvement

Understanding the software architecture of the quadcopter was not as easy as we had expected. The software is open source, yet the architectural documentation available from the community of experts familiar with the quadcopter is scarce. In the end, our software engineers had to reconstruct the architecture, identifying modules, interfaces, and operational dynamics from the open source code. Appendix E provides a top-level diagram of the reconstructed architecture.

As a team, we discussed the identified security risks and how they would be handled by the as-is software architecture. We then selected one risk, the ability of a malicious entity to send a successful “re-boot” command while the quadcopter is in flight, and identified an essential architectural mechanism that would mitigate the risk, authentication. We determined this was a feasible change to the architecture: the design of a new version of the MAVLink protocol allowed authentication, but it was not implemented in the existing architecture. Using the COCOMO II tool, we estimated the cost of building the software to implement authentication. Our team’s expert software engineer implemented the change, recording the effort required. We tested the quadcopter’s behavior before and after implementing the change. Before the change, the attack succeeded, causing the quadcopter to drop abruptly; after the change, the quadcopter was no longer susceptible to this attack. The cost and effort data are presented in Section 4.3.

⁸ STRIDE stands for Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Services, and Elevation of Privilege [Microsoft 2005, Howard 2006].

⁹ The categories and criteria focus on architecture mechanisms to be incorporated into the quadcopter, rather than on policy and procedural enablers of security.

The architectural change also mitigated several other security risks that were enabled by a lack of authentication. Note that selecting and properly implementing key, basic security mechanisms, such as strong authentication, can often mitigate multiple risks.

4.3 Cost Estimation Results

As described in Section 4.2, we proposed mitigating the risk that a malicious actor would issue a “reboot” command to the quadcopter in flight. The mitigation consisted of implementing authentication in MAVLink, on the quadcopter side only (not on the Ground Control Station side), in order to reject the unauthorized access and command. We estimated the cost of such a change using our COCOMO-based tool. A senior engineer then implemented the architectural change, monitoring effort.

Estimates. Tables 2 and 3 and Figure 7 show the estimated (via our COCOMO tool) costs for making architectural updates to the quadcopter software to resolve the threat of a malicious party issuing a “reboot” command while the quadcopter is in flight. Specifically, Table 2 shows the estimation results for years 1-8 and Table 3 shows the results for years 9-15 and total. (Note that in these tables “drone” was used in place of “quadcopter.”) Figure 7 plots the estimation results for the entire 15 years.

Table 2. Cost Estimation Results, first 8 years

Name	KSloc	\$K 1	\$K 2	\$K 3	\$K 4	\$K 5	\$K 6	\$K 7	\$K 8
Drone_OA_UPDT_1_DEV	1	4	0	0	0	0	0	0	0
Drone_OA_UPDT_1_DEV	1	4	4	4	4	4	4	4	4
Drone_OA_UPDT_1_MAINT	78.463	376	400	419	444	469	489	519	546
Drone_OA_UPDT_1_MAINT	78.463	376	776	1195	1639	2108	2597	3116	3662
Drone_OA_REAL_MAINT	78.319	376	400	418	443	468	488	518	545
Drone_OA_REAL_MAINT	78.319	376	776	1194	1637	2105	2593	3111	3656

Table 3. Cost Estimation Results, last 7 years and total

Name	KSloc	\$K 9	\$K10	\$K11	\$K12	\$K13	\$K14	\$K15	\$K Total
Drone_OA_UPDT_1_DEV	1	0	0	0	0	0	0	0	4
Drone_OA_UPDT_1_DEV	1	4	4	4	4	4	4	4	
Drone_OA_UPDT_1_MAINT	78.463	578	602	637	672	711	743	785	8390
Drone_OA_UPDT_1_MAINT	78.463	4240	4842	5479	6151	6862	7605	8390	
Drone_OA_REAL_MAINT	78.319	577	600	636	671	710	742	783	8375
Drone_OA_REAL_MAINT	78.319	4233	4833	5469	6140	6850	7592	8375	

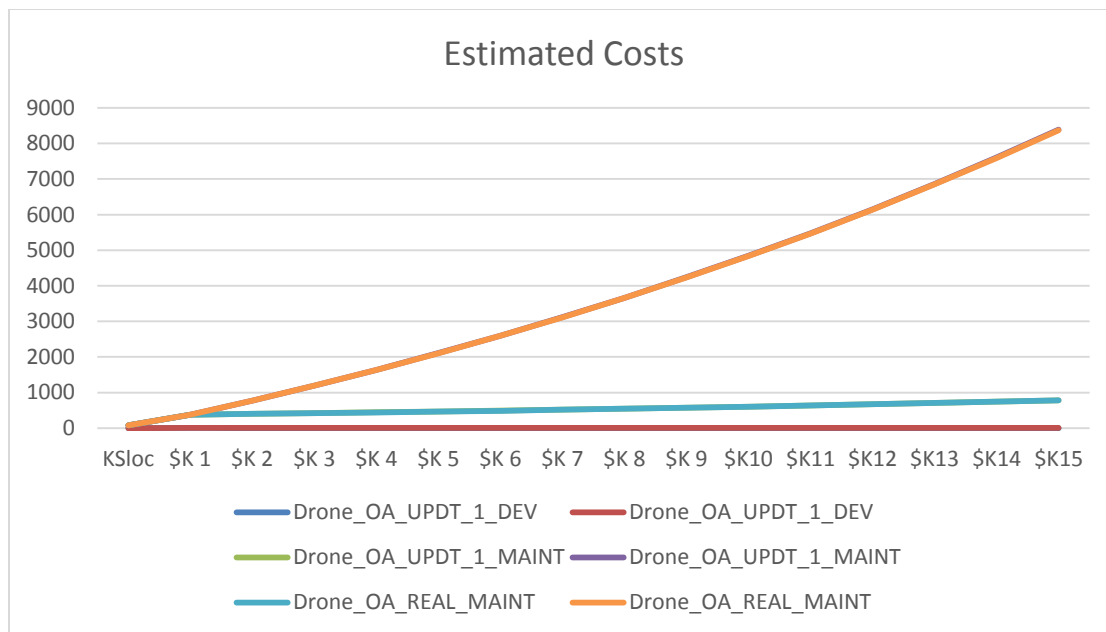


Figure 11. Estimation results for 15 years

Accuracy of Estimation of Effort: Validation Result. After the above estimates were created, one of our senior engineers implemented and tested a change called for in the estimation. Table 4 shows the software size (in SLOC) and effort (in hours) for two development efforts. While the total predicted value was 27.7 hours, this would have included full regression testing, documentation update, and other activities implied in COCOMO II. Thus, our engineer's actual effort for just the coding and validation was high; however, this was his only experience coding for this "project" so he had zero learning curve.

Table 4. Actual effort implementing change in drone software

Adding Authentication to MAVLink (only on drone side, not the GCS; includes only coding and testing effort)						
	SLOC			Effort (h)	SLOC/h	Predicted
	Before	After	Delta			
	1235	1242	7			
	12253	12390	137			
	13488	13632	144	10.3	14.0	
dataman						
mavlink						
TOTAL						
Partial imple- mentation			135	9.7		
TOTAL			279	20.0		
Modifying MAVLink router and MAVLink to log the IP of the sender of unauthorized messages						
	SLOC			Effort (h)	SLOC/h	Predicted
	Before	After	Delta			
	154124	154165	41			
	12390	12437	47			
	166514	166602	88	11.0	8.0	
mavlink router						
mavlink						
TOTAL						
Total (both)				31		27.7

5 Observations and Lessons Learned

The SWATT project has been a pioneering attempt to incorporate the effects of *software* design decisions into early system cost estimation efforts. The SWATT tool chain has been shown to be useful in analyzing the effects of alternative architectural approaches (e.g., for achieving qualities such as openness and security), on the estimated lifecycle costs of a system. The soundness of the approach of using empirical architectural parameters to refine the inputs to a cost model has been confirmed by one of the creators of COCOMO II, and our collaborating organization plans to use the SWATT cost estimation tool in future cost estimation analyses.

5.1 Caveat

A caveat must be mentioned regarding these results. While the results showed the feasibility of our approach on notional systems, the SWATT analyses have not yet been used on actual programs of record. The FY2016 work analyzed the architecture of a notional tank (fighting vehicle) using legacy cost data. The FY2017 work analyzed the open-source architecture of a quadcopter, not the preferred UAV, and the lack of architectural documentation hampered the effort. Using the tool chain on a program of record would validate the tool and would also help the program of record with an as-yet-unmet need.

The overall approach of connecting quantified ratings of software architecture qualities such as openness and security to a software cost prediction tool is both feasible and worthwhile.

5.2 Observations and lessons learned from the SWATT project

Here are some other observations and lessons learned from the SWATT project.

Intent of task

- Connecting open architecture aspects of a software architecture to a software cost prediction tool is both possible and useful.
- To date, no one else has mapped architectural decisions to software maintenance operations and cost.

Quality Attributes

- Operational quality attributes (e.g., performance, availability, or security) are not as simple to analyze as constructive quality attributes (e.g., modularity or layering). Operational quality attributes are not characterized by the software alone; additional information is required about how the operational system works in a particular context. Constructive quality attributes can be estimated when the first architecture drawing is available.
- There is a definite need to understand the trade-offs between security and other quality attributes, and the cost of building security into an architecture. However, checklists alone cannot ensure that systems behave in a secure manner. Security experts need to be involved in analyzing operational mission threads so they can understand the context and requirements for security relative to other requirements, particularly if someone wants to use the SWATT tool chain to estimate cost of a certain “expected” level of security.

Difference between Security and other Quality Attributes

- The original intention to create a security checklist for the avionics domain (and possibly others) proved to be a complex analytic task. Security as a quality attribute involves much more than just the structure of the architecture. We needed to posit mission threads, scenarios, attacks, vulnerabilities, and countermeasures. Terminology conflicts and different perspectives on security had to be untangled, sorted through, and managed.
- The approach taken to adapt the open architecture (OA) checklist for security was accepted by the developers of the OA checklist. Architecture specialists who attended a briefing on the approach agreed that it made sense to include architecture non-specific “concerns” and architecture-specific “security” because of the need for security investigations to start with some assumptions about architectural features.

Activities

- A nominal mission thread should be developed at a high level. Security analysis consist of a series of questions: (1) what kinds of attacks could occur in each step (we used STRIDE as a framework to identify attacks); (2) what vulnerabilities in the architecture could be exploited to enable the attack; and (3) what kinds of countermeasures could reduce the likelihood and consequences of the attack? These questions, which include architecture-specific countermeasures, are the basis for security questions.
- The open architecture checklist is used to assess each “key software component” in a software architecture. Previous SEI experience has shown that the ideal unit of assessment—the “sweet spot” for incorporating architectural considerations into cost estimates—is the Computer Software Configuration Item (CSCI). This implies that the software architecture has been determined at least to that level.

Tools

- Normally one performs a COCOMO estimate using available detailed information about the implementation. The SWATT project found a way to produce estimates without detailed implementation information, only architecture information.
- COCOMO II uses a large number of inputs for calibration, and it is not clear they all matter in the SWATT context. (The complexity factor in particular was complex: it needed a 5 row by 6 column table just to understand it.) The SWATT project focused on just a few that proved useful from an open architecture perspective. Other inputs were simply set to “Nominal.”
- STRIDE was useful as an initial, ready-made framework, but ideally we would have developed a more robust threat classification model derived from multiple sources.

Architecture Documentation

- There was no documentation of the quadcopter software architecture. It was open source software, with little effort expended in providing anything other than code. Even after requesting architecture information in open-source forums, SEI researchers were unable to get any more information. This is not an indictment of the code itself; the problem is that there is no accompanying documentation of structural elements that would have permitted reasoning about quality attributes. Thus to determine if the quadcopter architecture satisfied the security scenarios, it was necessary to analyze the code and recreate the architecture from the code interfaces. In addition, some of the

things a scenario asked for (e.g., an attacker issues a command to reboot the system) needed information below the level of architectural elements. SEI researchers had to examine the code itself to answer these questions.

General lessons learned

- A project that anticipates using data from actual military programs will have its results restricted even if none of the data actually used is from real programs. All of such a project's communication and outputs will need to go through the release review process.
- It is easiest to find and use publicly accessible data. For a variety of reasons, the team may not gain access to anticipated data from non-public sources. Collaborating projects may not provide anticipated data. Architects and other implementers may be wary of sharing architecture-level data. Concerns vary from classification to proprietary information disclosure. Even if agreements are in place, those agreements may change.
- Collaborators' priorities may change and as a result, they may defer completion of SEI work. Project teams need to reach an agreement with collaborators on suitable response times for review and approval, and on what data and information will and will not be acceptable for SEI to release. Working to ensure collaborators understand the rationale for deadlines and guidance on releasable content is also essential.

6 Future Work

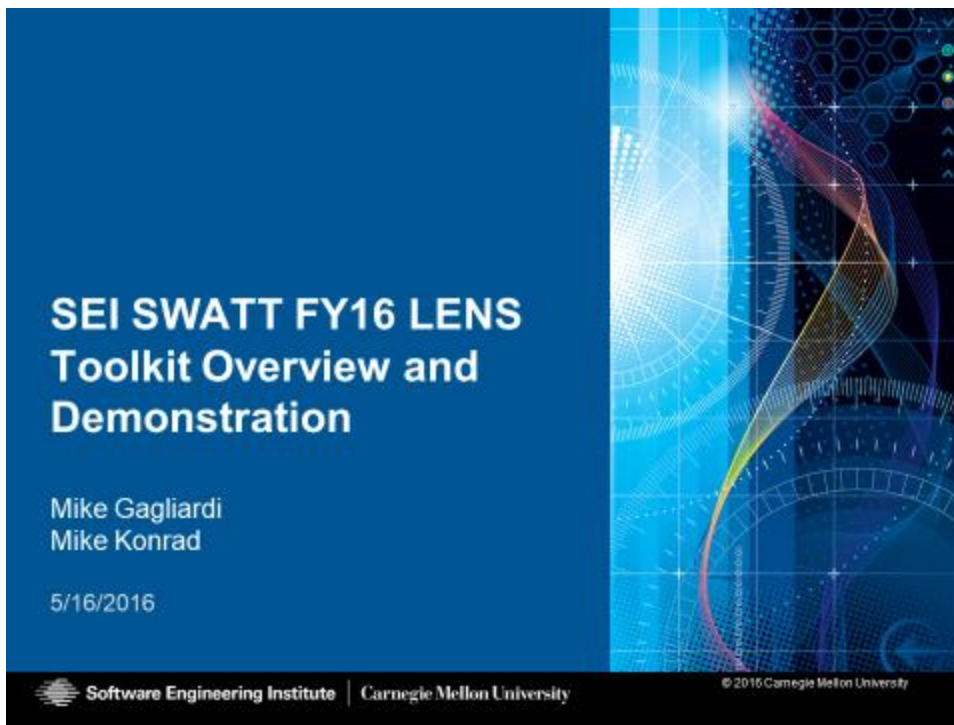
To make the SWATT toolset comprehensive, it should evaluate all software-related attributes of a candidate high-level software architecture. Its applicability across other domains will also need to be validated. These are considerable challenges.

A more pragmatic, near-term alternative is to continue to analyze security (a single quality attribute that has broad applicability) in order to evaluate other threats, vulnerabilities, and countermeasures, and determine approaches to both estimate architectural contributions to security and to fix architectures that lack adequate security. The SWATT project could also be applied for other vehicles that use avionics (e.g., a larger-scale drone instead of a quadcopter). With this kind of enhancement it would be possible, for example, to quantify the cost of implementing countermeasures or achieving minimum vulnerabilities on an actual military vehicle.

The work to date provides a basis for a general method for deriving a security checklist for a different domain. It could also be used to analyze tradeoffs between openness and security. In the near term, SEI:

- Should continue to work with our collaborating organization and its contractors, to help them implement our tools.
- Would like to analyze the cost/security tradeoffs when there are a number of identical or similar drones together (such as in a swarm). Would there be economies of scale, or would the complexity cause more problems than the configuration solves?
- Would like to look at the security and cost issues for a system that is part of a SoS.
- Would like to continue implementation work on the quadcopter, in order to reduce the likelihood and cost of exploitation of vulnerabilities for other threats.
- Would like to automate analysis given some initial inputs, rather than requiring senior engineers to review every step.

Appendix A. Demonstration slide set



Outline



Overview of FY16 toolkit and demos

Demo 1: SW cost estimation widget

Demo 2: Adding open architecture
considerations to SW cost
estimation

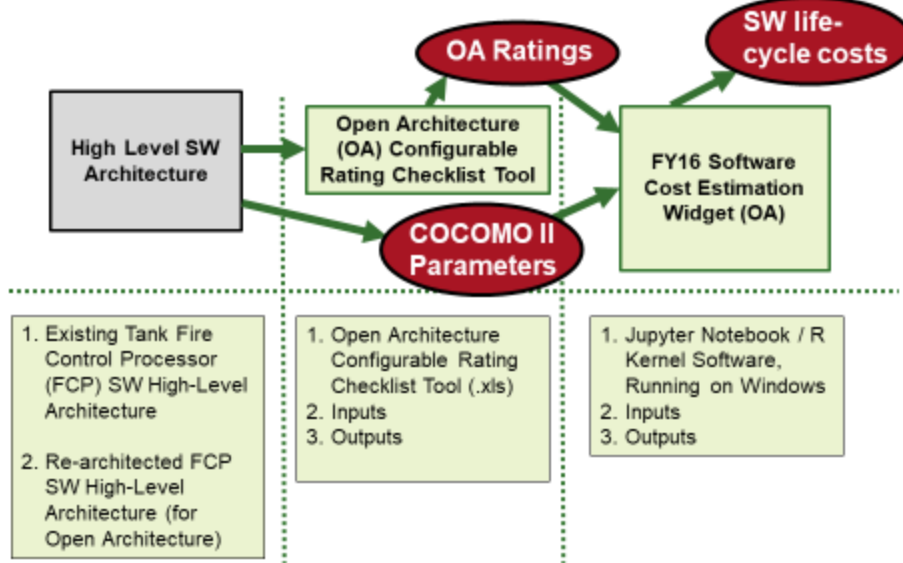
Demo 3: Adding alternative SW architecture
trajectory to SW cost estimation

Demo 4: Life-cycle cost comparison of two
SW architecture trajectories

Discussion:

- Q&A, issues, etc.
- Integration into client pipeline
- Potential pilot program

FY16 SWATT “Toolkit” Flow



Demo 1: SW Cost Estimation Widget

Existing Tank Fire Control Processing SW

Software Engineering Institute | Carnegie Mellon University

© 2018 Carnegie Mellon University

5

Example Architecture (Notional)

The example used is loosely based on architectures for infantry fighting vehicles.

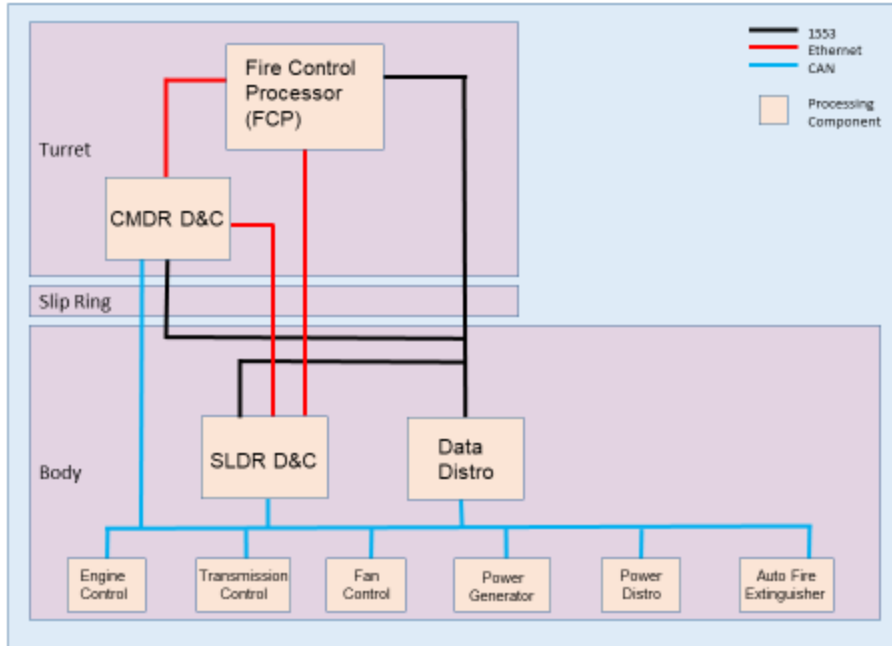


Software Engineering Institute | Carnegie Mellon University

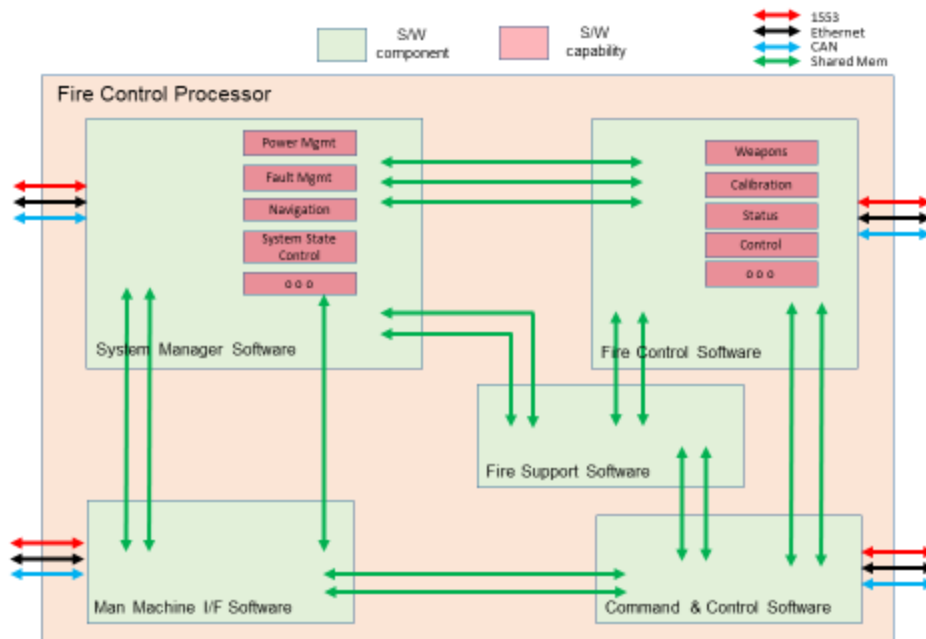
© 2018 Carnegie Mellon University

6

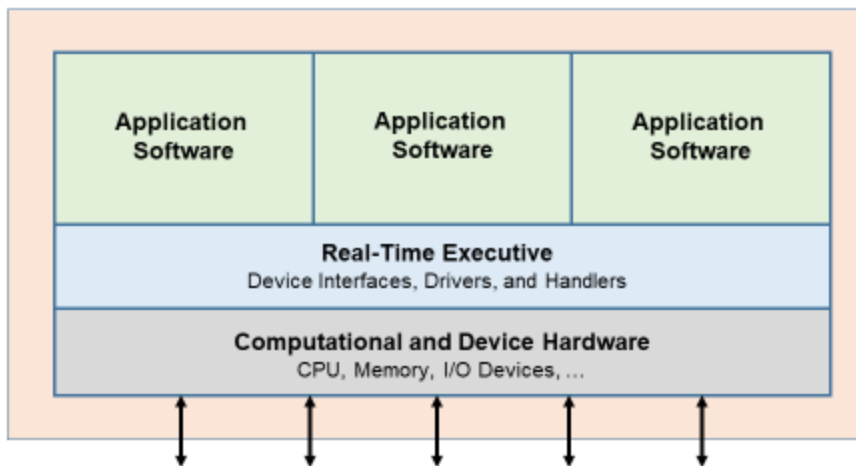
Existing System Architecture



Existing Fire Control Processor – High Level Software Functional Decomp



Existing Processing Component Layering Diagram



Now suppose...

A program office acknowledges that the existing FCP software O&M costs are onerous and they want to estimate the O&M costs for existing FCP software over the next 15 years. The program office can:

1. Execute the FY16 Software Cost Estimation Widget* for the FCP software, specifying a range of 15 years.

* The Widget can be used for any high-level software architecture (e.g. existing/legacy software, envisioned changes to existing/legacy software, newly developed software, candidate software architectures (pre MS-A), etc.)

FY16 SWATT SW Cost Estimation Widget



Software Engineering Institute | Carnegie Mellon University

© 2016 Carnegie Mellon University

11

Widget Implements COCOMO II* to Estimate Effort

The cost estimation model COCOMO II requires rating ~30 factors:

- **Software size:** effort spent in writing or modifying code
 - Measured or converted into thousands of lines of source code (KSLOC)
- **Scaling factors:** grow non-linearly with size (economies of scale)
 - Software size is raised to this power
 - Assumed to be project-wide
- **Cost drivers:** increase effort linearly ("effort multipliers")
 - Assumed to be component-specific (except for SCED)

Total effort = $A * (\text{Software_size}^B * \text{Scale_factors}) * \text{Cost_drivers}$

- A and B are constants set by standard or calibration
- Total cost and schedule duration are functions of Total effort.

*COCOMO II Model Definition Manual V2.1 (2000)



Software Engineering Institute | Carnegie Mellon University

© 2016 Carnegie Mellon University

12

Using COCOMO II in the Widget

- [1] $AAF = (0.4 * DM_i) + (0.3 * CM_i) + (0.3 * IM_i)$
- [2] $(AAF \leq 50) AAM = (AA_i + (AAF * (1 + (0.02 * SU_i * UNFM_i)))) / 100$
 $(AAF > 50) AAM = (AA_i + (AAF + (SU_i * UNFM_i))) / 100$
- [3] $SIZE_i = KSLOC_i * GRWTH_i * (1 - (AT_i / 100)) * AAM * (1 + (REVL_i / 100))$
- [4] $SIZE_{Aggregate} = \sum_{i=1}^n SIZE_i$
- [5] $E = B + (0.01 * (PREC + RESL + TEAM + PMAT + FLEX))$
- [6] $PM_{Basic} = A * (SIZE_{Aggregate})^E * SCED$
- [7] $PM_{Basic(i)} = PM_{Basic} * (SIZE_i / SIZE_{Aggregate})$
- [8] $PM_i = PM_{Basic(i)} * DATA * RUSE * DOCU * TIME * STOR * PVOL * ACAP * PCAP * PCON * APEX * PLEX * LTEX * TOOL * SITE * SCED * RELY * CPLX_i$
- [9] $PM_{Aggregate} = \sum_{i=1}^n PM_i$
- [10] $Cost = (PM_{Aggregate} * LABOR) / 1000$ (\$K per year, not in COCOMO II Model)

Green = Component based

Blue = Project based

Inputs

COCOMO II Project Parameters (Excel .csv file)

- Filename: "Tank_As_Is_NO_OA_Fixed_Input.csv"

OA1	OA2	OA3	OA4	PREC	RESL	TEAM	PMAT	GRWTH	AA	DM	CM	IM	SU	UNFM	REVL	PREC	RELY	CPLX	W	B	LABOR			
-1	-1	-1	-1	5.0	1.0	1.0	1.1	3.0	1.2	1	1.1	1.2	1.1	1.1	0.7	0.8	0.8	0.8	1	1	1	134	92	3500

COCOMO II Component Parameters (Excel .csv file)

- Filename: "Tank_As_Is_NO_OA_SW_Input.csv"

Name	KSLOC	OA1	OA2	OA3	OA4	AA	DM	CM	IM	SU	UNFM	REVL	PREC	RELY	CPLX	GRWTH	AT
FCP-SM	250	-1	-1	-1	-1	0	100	100	100	35	0.5	0	0	1	1	0.1	0
FCP-FC	250	-1	-1	-1	-1	0	100	100	100	35	0.5	0	0	1	1	0.1	0
FCP-FS	100	-1	-1	-1	-1	0	100	100	100	35	0.5	0	0	1	1	0.1	0
FCP-MMI	150	-1	-1	-1	-1	0	100	100	100	35	0.5	0	0	1	1	0.1	0
FCP-CBC	150	-1	-1	-1	-1	0	100	100	100	35	0.5	0	0	1	1	0.1	0
RT_EXEC	75	-1	-1	-1	-1	0	100	100	100	35	0.5	0	0	1	1	0.1	0

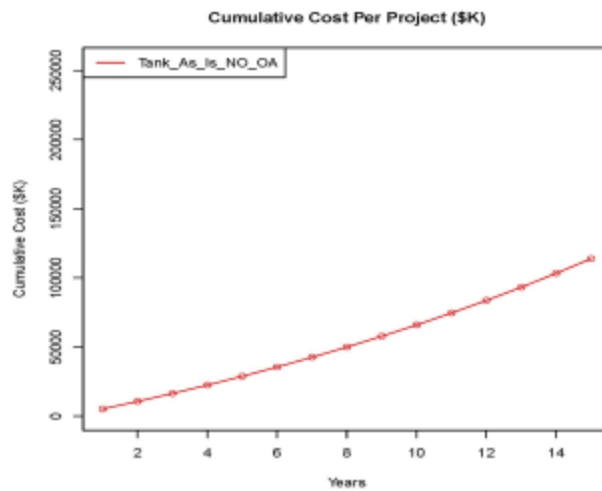
Execute FY16 Software Cost Estimation Widget

Output₁ Yearly SW Costs per Component (Excel .csv file)

Filename: "Tank_As_Is_Output.csv"

Name	KSloc	\$K 1	\$K 2	\$K 3	\$K 4	\$K 5	\$K 6	\$K 7	\$K 8	\$K 9	\$K 10	\$K 11	\$K 12	\$K 13	\$K 14	\$K 15	\$K Total
FCP-SM	150	890	997	985	1097	1090	1148	1208	1271	1337	1407	1480	1557	1638	1724	1814	19324
FCP-FC	250	1484	1561	1643	1728	1818	1913	2013	2118	2229	2345	2467	2596	2731	2874	3023	32548
FCP-FS	100	593	624	657	691	727	765	805	847	891	938	987	1038	1092	1149	1209	13013
FCP-MM	150	890	997	985	1097	1090	1148	1208	1271	1337	1407	1480	1557	1638	1724	1814	19324
FCP-C&C	150	890	997	985	1097	1090	1148	1208	1271	1337	1407	1480	1557	1638	1724	1814	19324
RT_EXRC	75	445	481	492	518	545	574	604	635	668	703	740	778	819	862	907	9758
Totals	875	5192	5464	5747	6048	6363	6686	7046	7413	7799	8207	8634	9083	9556	10057	10581	113886
Cumulative \$K		5192	10656	16403	22451	28814	35500	42556	49969	57768	65975	74609	83692	93248	103305	113886	

Output₂ Cumulative Yearly SW Costs per Project



SW Cost Estimation Widget Implementation

- Jupyter Notebook (v 4.0.4) (from Collaborator)
- R Kernel (from Collaborator)
- Currently Executing on MS Windows
- Modular
 - can easily replace the COCOMO II cost estimation functions
- File-Based Inputs and Outputs
 - Excel .csv files
- Approx. 200 SLOC (R)
 - 5 Functions, 1 Main

Demo 2: Adding Open Architecture Considerations to SW Cost Estimation

existing Tank Fire Control Processing SW



Software Engineering Institute | Carnegie Mellon University

© 2018 Carnegie Mellon University

19

Now suppose...

A program office acknowledges that the existing FCP software O&M costs are un-sustainable and they do not currently have visibility into the factors associated with the construction of the FCP software that have significant contributions to the O&M costs. Decides to investigate and do the following:

1. Baseline the existing FCP SW architecture risks with respect to Open Architecture
2. Account for the FCP SW Open Architecture risks in the cost estimations over the next 15 years.



Software Engineering Institute | Carnegie Mellon University

© 2018 Carnegie Mellon University

20

Open Architecture Configurable Rating Checklist Tool



Software Engineering Institute | Carnegie Mellon University

© 2018 Carnegie Mellon University

21

Open Architecture Configurable Rating Checklist Tool

The Configurable Rating Checklist is intended to be used by an objective observer* to help assess the suitability of a proposed architecture for achieving specific objectives in the context of the program. Includes 7 +/- 2 questions about each attribute and combines answers into a ranking.

The assessment is applied to each "key software component" in a software architecture.

- **Inputs:** A high-level software architecture identifying key software components, their relationships, and properties. Explanations and justifications from software architects. Agreed upon rating for each criteria (0-5, 5 being highest) per attribute, per component
- **Outputs:** Automatically produces a weighted score (0-5, 5 being highest) for each key attribute (Modularity, Layering, Interface Standards, and Open and Accessible Standards) for an assessed software component.

The Configurable Rating Checklist has been completed with feedback and support from the customer.

* The Configurable Rating Checklist is not intended as a self-assessment tool for program offices.

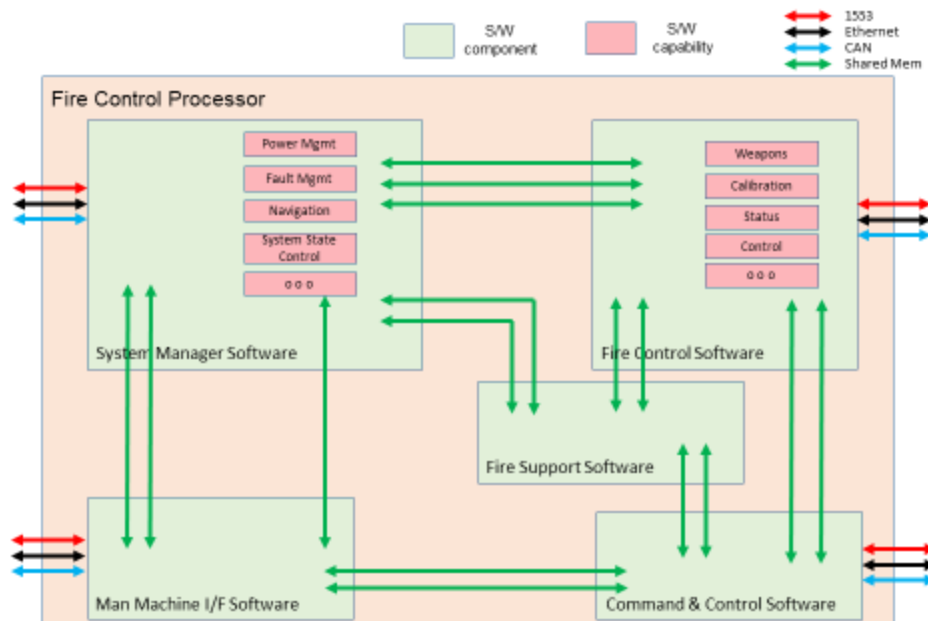


Software Engineering Institute | Carnegie Mellon University

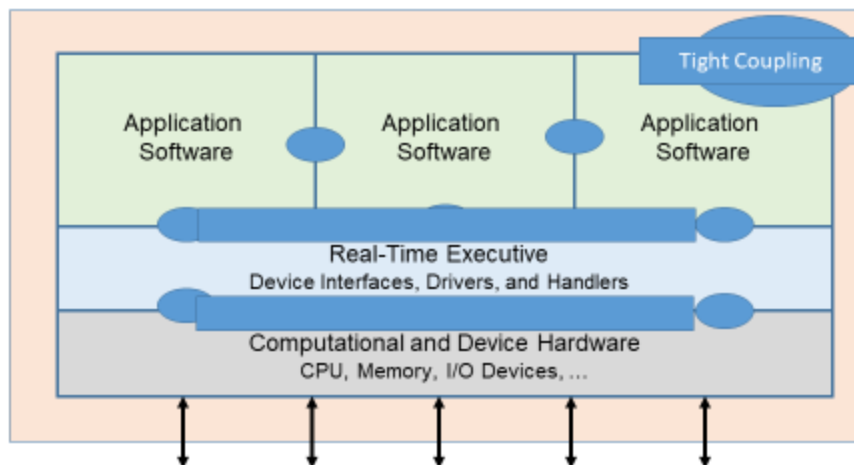
© 2018 Carnegie Mellon University

22

Existing Fire Control Processor – High Level Software Functional Decomp



Existing Processing Component Layering Diagram



Existing Software Architecture - Fire Control Processor

Characteristics:

- **Modularity**

- Fire Control Processor, five major software components:
 - o components are not coherent; poor separation of concern
 - o components do not follow good programming practices
 - o components contain "dead" code and data
 - o components tightly coupled to each other

- **Layering**

- Real-Time Executive (RT-Exec) layer between applications and hardware
 - o proprietary
- Applications tightly-coupled to proprietary RT-Exec

- **Interfaces**

- Proprietary, non-standard

- **Documentation**

- Code level
- Design documentation out of date

Existing system Open Architecture ratings

From Configurable Ratings Checklist

Attribute	Weighted Attribute score (0-5)
Modularity (OA1)	0.77
Layering (OA2)	0.95
Interface Standards (OA3)	0.44

Using COCOMO II in the Widget

OA 3

- [1] $AAF = (0.4 * DM_i) + (0.3 * CM_i) + (0.3 * IM_i)$
- [2] $(AAF \leq 50) AAM = (AA_i + (AAF * (1 + (0.02 * SU_i * UNFM_i)))) / 100$
 $(AAF > 50) AAM = (AA_i + (AAF * (SU_i * UNFM_i))) / 100$
- [3] $SIZE_i = KSLOC_i * GRWTH_i * (1 - (AT_i / 100)) * AAM * (1 + (REVL_i / 100))$
- [4] $SIZE_{Aggregate} = \sum_{i=1}^n SIZE_i$
- [5] $E = B + (0.01 * (PREC + RESL + TEAM + PMAT + FLEX))$
- [6] $PM_{Basic} = A * (SIZE_{Aggregate})^E * SCED$
- [7] $PM_{Basic(i)} = PM_{Basic} * (SIZE_i / SIZE_{Aggregate})$
- [8] $PM_i = PM_{Basic(i)} * DATA * RUSE * DOCU * TIME * STOR * PVOL * ACAP * PCAP * PCON * APEX * PLEX * LTEX * TOOL * SITE * SCED * RELY * CPLX_i$
- [9] $PM_{Aggregate} = \sum_{i=1}^n PM_i$
- [10] $Cost = (PM_{Aggregate} * LABOR) / 1000$ (\$K per year, not in COCOMO II Model)

Green = Component based

Blue = Project based

OA1 (Modularity), OA2 (Layering) and OA3 (Interface Standards) Mapping to COCOMO II

No modifications to the COCOMO II Model. No changes to the COCOMO II range of values for the impacted "component" terms (SU, UNFM, CPLX, and FLEX). The OA1, OA2, and OA3 ratings are used to "fine-tune" the terms to more accurately reflect the OA characteristics of the component.

OA1	SU_MAX	SU_MIN	UNFM_MIN	UNFM_MAX	CPLX_MIN	CPLX_MAX	FLEX_MIN	FLEX_MAX
5	25	10	0.17	0	2	1.41		
4	30	15	0.33	0.17	3	2		
3	35	20	0.5	0.33	4	3		
2	40	30	0.67	0.5	5	4		
1	45	35	0.83	0.67	6	5		
0	50	40	1	0.83	7.07	6		

OA2	SU_MAX	SU_MIN	UNFM_MIN	UNFM_MAX	CPLX_MIN	CPLX_MAX	FLEX_MIN	FLEX_MAX
5	25	10	0.17	0	0.87	0.73	2	1.41
4	30	15	0.33	0.17	1	0.87	3	2
3	35	20	0.5	0.33	1.17	1	4	3
2	40	30	0.67	0.5	1.34	1.17	5	4
1	45	35	0.83	0.67	1.74	1.34	6	5
0	50	40	1	0.83	1.74	1.74	7.07	6

OA3	SU_MAX	SU_MIN	UNFM_MIN	UNFM_MAX	CPLX_MIN	CPLX_MAX	FLEX_MIN	FLEX_MAX
5	25	10	0.17	0	0.87	0.73	5.07	2
4	30	15	0.33	0.17	1	0.87	4.6	1.6
3	35	20	0.5	0.33	1.17	1	4.2	1.2
2	40	30	0.67	0.5	1.34	1.17	3.8	0.8
1	45	35	0.83	0.67	1.74	1.34	3.4	0.4
0	50	40	1	0.83	1.74	1.74	3	0

Inputs

COCOMO II Project Parameters (Excel .csv file)

- Filename: "Tank_As_Is_Fixed_Input.csv"

OA1	OA2	OA3	OA4	FLD	PREC	REL	TEAM	PMAT	DATA	RLSE	DOCU	TIME	STOR	PIVL	ACAP	PCAP	PCON	APRI	PREL	STD	TOOL	SITE	SCD	RELY	CPUL	A	B	LABOR
1	1	0	0	0	0.77	0.95	1.40	1.1	3.02	1.28	1	1.0	1.28	1.0	1.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1	1	1	1.94	0.0	2900

OA Ratings

COCOMO II Component Parameters (Excel .csv file)

- Filename: "Tank_As_Is_SW_Input.csv"

Name	KSLOC	OA1	OA2	OA3	OA4	AA	DM	CM	IM	SU	UNFM	REVL	PREC	RELY	CPUL	GROWTH	AT
FCP-SM	150	0.77	0.95	0.44	-1	0	100	100	100	35	0.5	0	0	1	1	0.1	0
FCP-FC	250	0.77	0.95	0.44	-1	0	100	100	100	35	0.5	0	0	1	1	0.1	0
FCP-FS	100	0.77	0.95	0.44	-1	0	100	100	100	35	0.5	0	0	1	1	0.1	0
FCP-MMI	150	0.77	0.95	0.44	-1	0	100	100	100	35	0.5	0	0	1	1	0.1	0
FCP-C&C	150	0.77	0.95	0.44	-1	0	100	100	100	35	0.5	0	0	1	1	0.1	0
RT_EXEC	75	0.77	0.95	0.44	-1	0	100	100	100	35	0.5	0	0	1	1	0.1	0

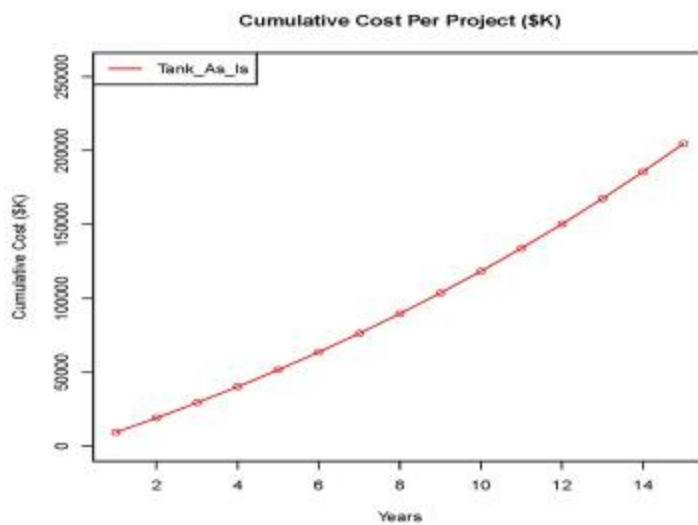
Output₁

Yearly Estimated SW Costs per Component (Excel .csv file)

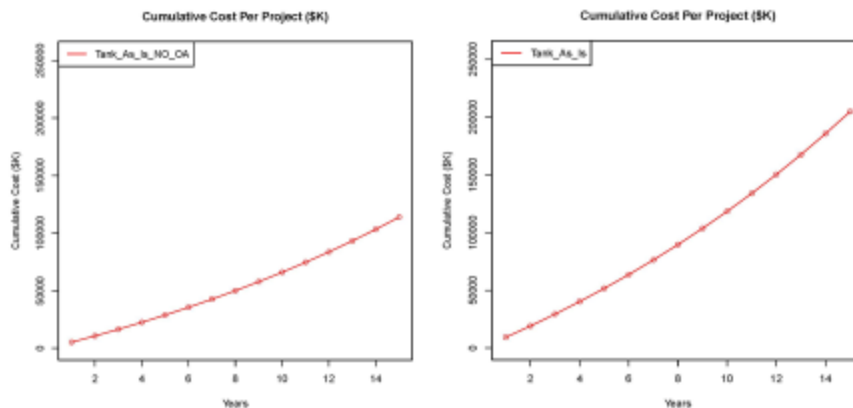
Filename: "Tank_As_Is_Output.csv"

Name	KSLOC	\$K.1	\$K.2	\$K.3	\$K.4	\$K.5	\$K.6	\$K.7	\$K.8	\$K.9	\$K.10	\$K.11	\$K.12	\$K.13	\$K.14	\$K.15	\$K.Total
FCP-SM	150	1591	1675	1764	1857	1955	2059	2168	2283	2403	2531	2665	2806	2954	3110	3275	35096
FCP-FC	250	2652	2792	2940	3095	3259	3432	3613	3805	4006	4218	4441	4676	4924	5184	5459	58496
FCP-FS	100	1060	1116	1176	1238	1303	1372	1445	1522	1602	1687	1776	1870	1969	2073	2183	23092
FCP-MMI	150	1591	1675	1764	1857	1955	2059	2168	2283	2403	2531	2665	2806	2954	3110	3275	35096
FCP-C&C	150	1591	1675	1764	1857	1955	2059	2168	2283	2403	2531	2665	2806	2954	3110	3275	35096
RT_EXEC	75	795	837	882	928	977	1029	1084	1141	1201	1265	1332	1403	1477	1555	1637	17543
Totals	875	9280	9770	10290	10832	11404	12010	12646	13317	14018	14763	15544	16367	17232	18142	19104	204719
Cumulative \$K		9280	19050	29340	40172	51576	63586	76232	89549	103567	118330	133874	150241	167473	185615	204719	

Output₂ Cumulative Yearly SW Costs per Project



Comparing SW Cost Estimates – Demo 1 & Demo 2



Demo 2 SW Cost Estimates (accounting for OA risks)
resulted in ~\$90M additional O&M costs after 15 years.

Demo 3: Adding Alternative SW Architecture Trajectory to SW Cost Estimation

Now suppose...

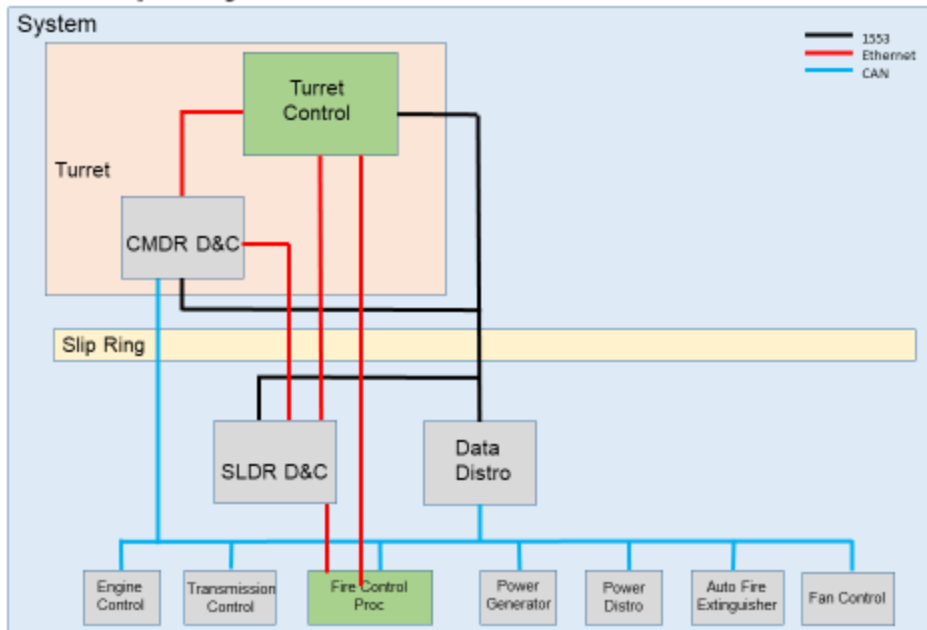
Program office acknowledges that the existing FCP software O&M costs are unsustainable to the point that alternative architecture approaches need to be investigated. Decides to investigate:

1. Re-architecting the FCP software to take advantage of expected open architecture benefits
2. Re-allocating the Fire Control Processing (FCP) software out of the turret and into the main processing unit of the vehicle
3. Estimate SW development and maintenance costs

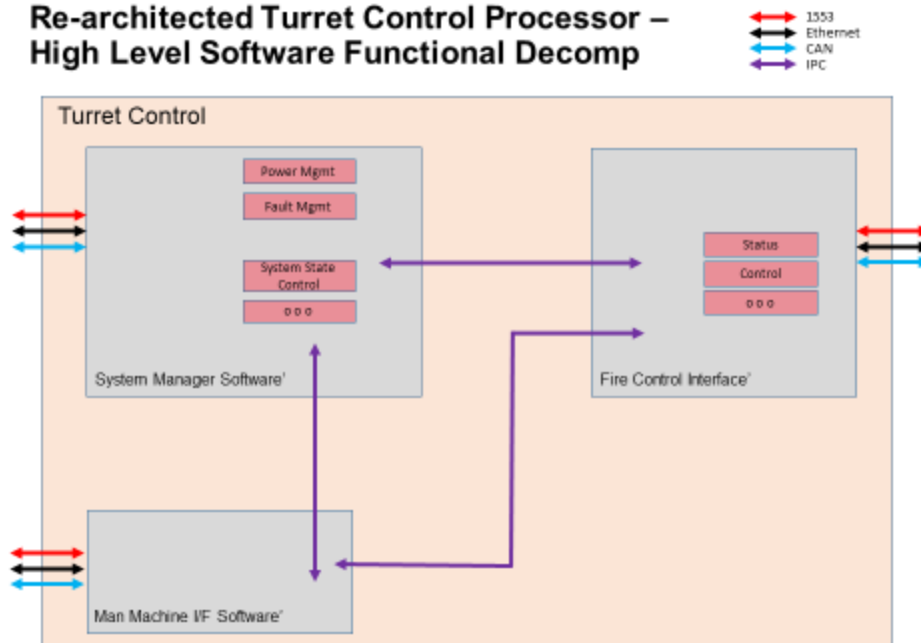
Expected benefits:

- Save on O&M costs for future FCP and turret upgrades
- Reduce regression testing and certification of the turret for FCP SW changes
- Eliminate vendor lock with turret vendor
- Reduce Field Service Engineering costs

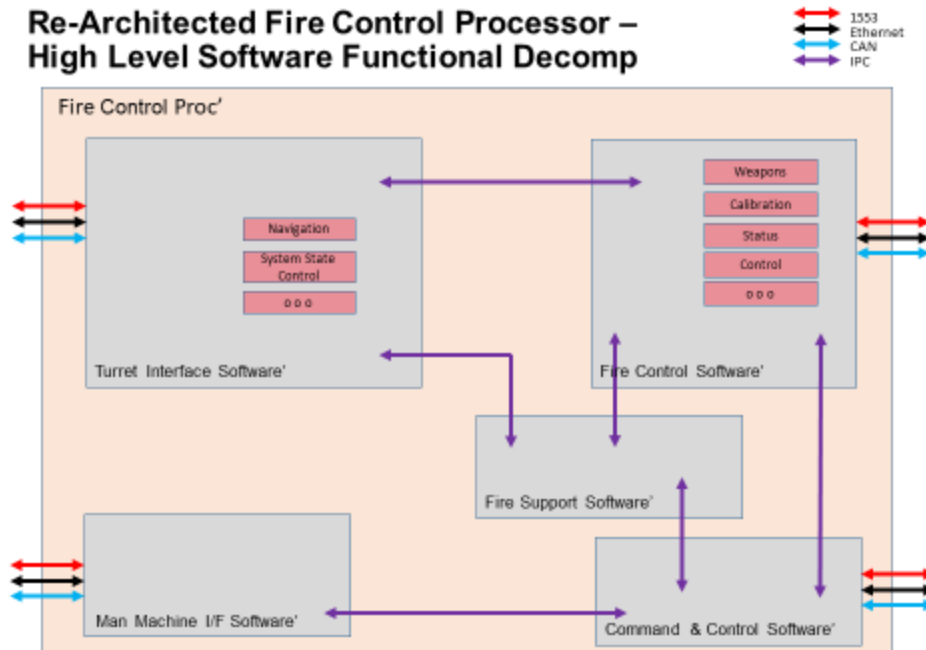
Example System Architecture²



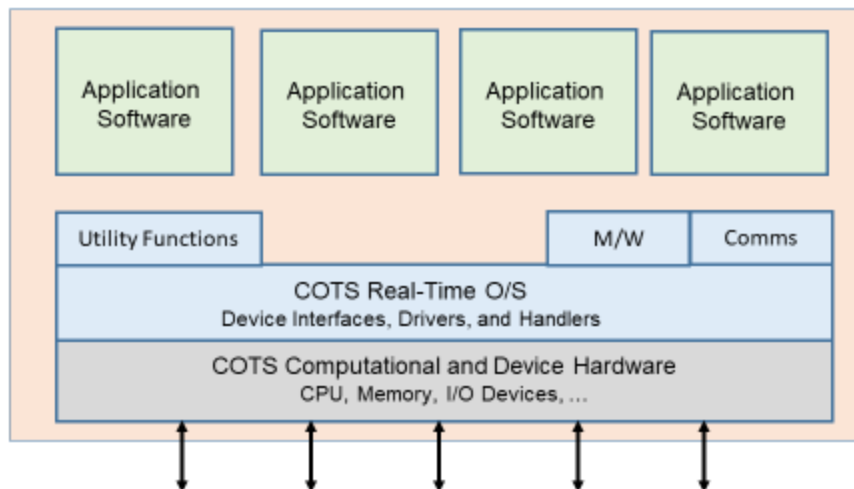
Re-architected Turret Control Processor – High Level Software Functional Decomp



Re-Architected Fire Control Processor – High Level Software Functional Decomp



Re-Architected Processing Component Layering Diagram



Re-Architected Software Architecture - Fire Control Processor

Characteristics:

- **Modularity**
 - Fire Control Processor, five major software components:
 - Components are coherent; good separation of concern
 - Components follow good programming practices
 - Components loosely coupled to each other
 - Turret Control Processor, three major software components:
 - Components are coherent; good separation of concern
 - Components follow good programming practices
 - Components loosely coupled to each other
- **Layering**
 - COTS Real-Time O/S Layer
 - Open Systems Utility Functions Layer
 - COTS Communication Layer
 - Applications loosely coupled
- **Interfaces**
 - Open standard based
- **Documentation**
 - Software architecture documented and conforms to implementation

Comparison of Open Architecture ratings

From Configurable Ratings Checklist

Attribute	Weighted Attribute score (0-5)	
	Existing	Rearchitected
Modularity (OA1)	0.77	4.29
Layering (OA2)	0.95	4.56
Interface Standards (OA3)	0.44	4.22

Inputs₁

COCOMO II Project Parameters (Excel .csv file)

- Filename: "Tank_OA_Endstate_Dev_Fixed_Input.csv"
- Filename: "Tank_OA_Endstate_Maint_Fixed_Input.csv"

OA1	OA2	OA3	OA4	PREC	PREC	RES	TEAM	PMAT	DATA	RESE	DOCU	TIME	STOR	PHIL	ACAP	PCAP	PCON	APEN	PLEN	STEN	TOOL	SITE	SCSD	RELY	CPLX	A	B	LABOR
4	4	4	-1	5.07	2.40	1.41	11	3.12	1.10	1.0	1.23	1.25	1.17	1.3	0.71	0.76	0.81	0.81	0.85	0.84	0.70	0.61	1	1	1	2.94	0.91	2500

OA1	OA2	OA3	OA4	PREC	PREC	RES	TEAM	PMAT	DATA	RESE	DOCU	TIME	STOR	PHIL	ACAP	PCAP	PCON	APEN	PLEN	STEN	TOOL	SITE	SCSD	RELY	CPLX	A	B	LABOR
4	4	4	-1	5.07	2.40	1.41	11	3.12	1.10	1	1.23	1.25	1.17	1.3	0.71	0.76	0.81	0.81	0.85	0.84	0.70	0.61	1	1	1	2.94	0.91	2500

Inputs₂

COCOMO II Component Parameters (Excel .csv file)

- Filename: "Tank_OA_Endstate_Dev_SW_Input.csv"
- Filename: "Tank_OA_Endstate_Maint_SW_Input.csv"

Name	KSLOC	OA1	OA2	OA3	OA4	AA	DM	CM	RM	SU	UNRM	REVL	PREC	RELY	CPLX	GRWTH	AT
TC-SM	50	4.3	4.6	4.2	-1	0	100	300	300	30	0.5	20	2.48	1	1	1	0
TC-FC	125	4.3	4.6	4.2	-1	0	100	300	300	30	0.5	20	2.48	1	1	1	0
TC-MWB	50	4.3	4.6	4.2	-1	0	100	300	300	30	0.5	20	2.48	1	1	1	0
FOP-TI	50	4.3	4.6	4.2	-1	0	100	300	300	30	0.5	20	2.48	1	1	1	0
FOP-FC	125	4.3	4.6	4.2	-1	0	100	300	300	30	0.5	20	2.48	1	1	1	0
FOP-FS	40	4.3	4.6	4.2	-1	0	100	300	300	30	0.5	20	2.48	1	1	1	0
FOP-MWB	75	4.3	4.6	4.2	-1	0	100	300	300	30	0.5	20	2.48	1	1	1	0
FOP-C&C	75	4.3	4.6	4.2	-1	0	100	300	300	30	0.5	20	2.48	1	1	1	0
NA/W Layer	10	4.3	4.6	4.2	-1	0	100	300	300	30	0.5	20	2.48	1	1	1	0
O/S Layer	50	4.3	4.6	4.2	-1	0	100	300	300	30	0.5	20	2.48	1	1	1	0
Utility Lay	10	4.3	4.6	4.2	-1	0	100	300	300	30	0.5	20	2.48	1	1	1	0
Comms Lt	25	4.3	4.6	4.2	-1	0	100	300	300	30	0.5	20	2.48	1	1	1	0

Name	KSLOC	OA1	OA2	OA3	OA4	AA	DM	CM	RM	SU	UNRM	REVL	PREC	RELY	CPLX	GRWTH	AT
TC-SM	50	4.3	4.6	4.2	-1	0	100	300	300	30	0.5	0	0	1	1	0.1	0
TC-FC	125	4.3	4.6	4.2	-1	0	100	300	300	30	0.5	0	0	1	1	0.1	0
TC-MWB	50	4.3	4.6	4.2	-1	0	100	300	300	30	0.5	0	0	1	1	0.1	0
FOP-TI	50	4.3	4.6	4.2	-1	0	100	300	300	30	0.5	0	0	1	1	0.1	0
FOP-FC	125	4.3	4.6	4.2	-1	0	100	300	300	30	0.5	0	0	1	1	0.1	0
FOP-FS	40	4.3	4.6	4.2	-1	0	100	300	300	30	0.5	0	0	1	1	0.1	0
FOP-MWB	75	4.3	4.6	4.2	-1	0	100	300	300	30	0.5	0	0	1	1	0.1	0
FOP-C&C	75	4.3	4.6	4.2	-1	0	100	300	300	30	0.5	0	0	1	1	0.1	0
NA/W Layer	10	4.3	4.6	4.2	-1	0	100	300	300	30	0.5	0	0	1	1	0.1	0
O/S Layer	50	4.3	4.6	4.2	-1	0	100	300	300	30	0.5	0	0	1	1	0.1	0
Utility Lay	10	4.3	4.6	4.2	-1	0	100	300	300	30	0.5	0	0	1	1	0.1	0
Comms Lt	25	4.3	4.6	4.2	-1	0	100	300	300	30	0.5	0	0	1	1	0.1	0

Output₁ Yearly SW Costs per Component₁

Filename: "Tank_OA_Endstate_Dev_Output.csv"

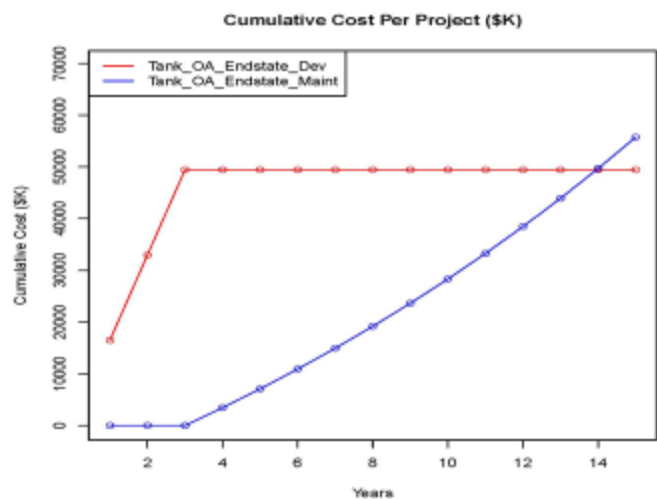
Name	KSloc	\$K.1	\$K.2	\$K.3	\$K.4	\$K.5	\$K.6	\$K.7	\$K.8	\$K.9	\$K.10	\$K.11	\$K.12	\$K.13	\$K.14	\$K.15	\$K.Total
TC-SM	50	1182	1182	1182	0	0	0	0	0	0	0	0	0	0	0	0	3546
TC-FC	125	3030	3030	3030	0	0	0	0	0	0	0	0	0	0	0	0	9090
TC-MMI	50	1182	1182	1182	0	0	0	0	0	0	0	0	0	0	0	0	3546
FCP-TI	50	1182	1182	1182	0	0	0	0	0	0	0	0	0	0	0	0	3546
FCP-FC	125	3030	3030	3030	0	0	0	0	0	0	0	0	0	0	0	0	9090
FCP-FS	40	960	960	960	0	0	0	0	0	0	0	0	0	0	0	0	2880
FCP-MMI	75	1847	1847	1847	0	0	0	0	0	0	0	0	0	0	0	0	5541
FCP-CMC	75	1847	1847	1847	0	0	0	0	0	0	0	0	0	0	0	0	5541
M/W Layer	10	221	221	221	0	0	0	0	0	0	0	0	0	0	0	0	663
O/S Layer	50	1182	1182	1182	0	0	0	0	0	0	0	0	0	0	0	0	3546
Utility Lay	10	221	221	221	0	0	0	0	0	0	0	0	0	0	0	0	663
Comms La	25	591	591	591	0	0	0	0	0	0	0	0	0	0	0	0	1773
Totals	685	16475	16475	16475	0	0	0	0	0	0	0	0	0	0	0	0	49425
Cumulative \$K		16475	32950	49425	49425	49425	49425	49425	49425	49425	49425	49425	49425	49425	49425	49425	

Output₂ Yearly SW Costs per Component₂

Filename: "Tank_OA_Endstate_Maint_Output.csv"

Name	KSloc	\$K.1	\$K.2	\$K.3	\$K.4	\$K.5	\$K.6	\$K.7	\$K.8	\$K.9	\$K.10	\$K.11	\$K.12	\$K.13	\$K.14	\$K.15	\$K.Total
TC-SM	50	0	0	0	252	285	279	294	309	325	342	360	379	399	420	442	4066
TC-FC	125	0	0	0	631	664	699	735	773	814	856	901	948	998	1050	1105	10174
TC-MMI	50	0	0	0	252	285	279	294	309	325	342	360	379	399	420	442	4066
FCP-TI	50	0	0	0	252	285	279	294	309	325	342	360	379	399	420	442	4066
FCP-FC	125	0	0	0	631	664	699	735	773	814	856	901	948	998	1050	1105	10174
FCP-FS	40	0	0	0	202	212	223	235	247	260	274	288	303	319	336	353	3252
FCP-MMI	75	0	0	0	378	398	419	441	464	488	514	540	569	598	630	663	6102
FCP-CMC	75	0	0	0	378	398	419	441	464	488	514	540	569	598	630	663	6102
M/W Layer	10	0	0	0	50	53	55	58	61	65	68	72	75	79	84	88	808
O/S Layer	50	0	0	0	252	285	279	294	309	325	342	360	379	399	420	442	4066
Utility Lay	10	0	0	0	50	53	55	58	61	65	68	72	75	79	84	88	808
Comms La	25	0	0	0	126	132	139	147	154	162	171	180	189	199	210	221	2030
Totals	685	0	0	0	3454	3634	3824	4026	4233	4456	4689	4934	5192	5464	5754	6054	55714
Cumulative \$K		0	0	0	3454	7088	10912	14938	19171	23627	28316	33250	38442	43906	49660	55714	

Output₃
Cumulative Yearly SW Costs per Project



Demo 4: Life-cycle Cost Comparison of Two SW Architecture Trajectories



Software Engineering Institute | Carnegie Mellon University

© 2018 Carnegie Mellon University

46

Trajectories

A trajectories script file is an input to the FY16 Widget. It contains a number of user specified projects, containing start and end years. The FY16 Widget generates cumulative, yearly SW estimates for each trajectory and plots the data for comparison purposes.

Filename: "Demo_4_Script.csv"

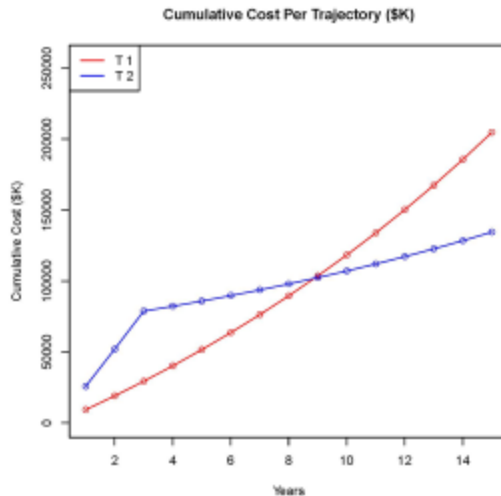
Input_File	Mode	Start_Year	Years	T1_Start	T1_Years	T2_Start	T2_Years	T3_Start	T3_Years
Tank_As_Is	M	1	15	1	15	1	3	-1	-1
Tank_OA_Endstate_Dev	D	1	3	-1	-1	1	3	-1	-1
Tank_OA_Endstate_Maint	M	4	12	-1	-1	4	12	-1	-1

In this example, we compare two architecture trajectories:

- T1:
 - Continue with maintenance of existing FCP SW for 15 years
- T2:
 - Continue with maintenance of existing FCP SW for 3 years
 - Re-architect for OA and implement FCP SW for 3 years
 - Cut-over to "Re-architected" FCP SW starting year 4 and maintain it for 12 years.

Output

Cumulative Yearly SW Costs per Trajectory



Trajectory 2 resulted in cumulative savings starting in year 9, resulting in ~\$75M in savings by year 15

Back-Up Slides

Factors COCOMO II Uses for Software Size

For **new code** (or reused code only needing some glue) estimate size by:

- Employing divide-and-conquer or historical analogy
- Reference to a standard for counting KSLOC

For **code that will be adapted**, estimate *Equivalent* KSLOC (EKSLOC) by considering these factors (6):

- How much design (or code) needs to be modified (DM or CM)
- How much additional integration test is required (IM)
- How easy it is to assess whether the code is appropriate for use (AA)
- How well structured the software is, making it easier to modify (SU)
- How familiar the adapter is with the code (UNFM)

Finally, **adjust** (multiply) to reflect percent of annual change due to:

- Requirements evolution and volatility during development (REVL)
- Change to the code base during maintenance (MCF)

(Abbreviation in parentheses is how the factor is referred to in COCOMO II equation.)

COCOMO II Scaling Factors

Grow non-linearly with size (reflect an **economy of scale**).

- Represented in the exponent of `Software_Size`

Have **qualitative rating levels** that express the impact on effort, ranging from **Very Low** to **Extra High**

- COCOMO II assigns a value (from 0 to ~8) to each qualitative rating.
- **Extra High** is set to 0.00 (meaning no impact on effort).
- Other rating levels have positive values (meaning effort grows non-linearly).

Rate your project with respect to each scaling factor (5):

- Precedented-ness (PREC)
- Development Flexibility (FLEX)
- [Uncertainty in] Architecture [is addressed through] Risk Resolution (RESL)
- Team cohesion (TEAM)
- Process maturity (PMAT)

COCOMO II Cost Drivers

Grow linearly with size (and thus are sometimes called “effort multipliers”)

Have **qualitative rating levels** that express the impact on effort, ranging from **Very Low** to **Very High** (goes up to **Extra High** in some cases)

- COCOMO II assigns a value (from ~0.7 to ~1.4) to each qualitative rating.
- **Nominal** rating level is set to 1.00 (meaning no impact on effort).

Rate each major component with respect to each product, development-environment, and personnel-related cost driver (17):

- An example cost driver: Product Complexity (CPLX)
 - Rating this factor requires integrating ratings of five sub-factors reflecting the source of complexity: control, computation, device-dependence, data management, and user interface management.
- An example application of cost driver Required Software Reliability (RELY):
 - If your project will develop software that controls an airplane's flight, you would set this cost driver to Very High. That rating corresponds to an effort multiplier of 1.26, meaning that your project will require 26% more effort than a typical software project.

Contact Information

Mike Gagliardi

Principal Engineer
Software Engineering Institute
Office: 412-268-7738
Cell: 412-926-5367
Email: mjg@sei.cmu.edu

Mike Konrad

Principal Engineer
Software Engineering Institute
Office: 412-268-5813
Email: mdk@sei.cmu.edu



Appendix B. Software package: Cost estimation using COCOMO

SEI SWATT FY16 Toolkit Software – README

Overview

The SEI developed a software program (“SEI FY16 SWATT Toolkit Software”, referred to as “Toolkit Software”) as part of the SEI FY16 LENS research project (“SEI FY16 SWATT Toolkit”, referred to as “SWATT Toolkit”) which provides early life-cycle cost software estimation (development and maintenance) of high-level software architectures. In addition, the SWATT Toolkit assesses the “openness” of high-level software architectures with respect to three main architectural attributes (modularity, layering, and interface standards) and accounts for the assessment results in the full life-cycle cost estimations. The “openness” assessment is not included in the Toolkit Software and is a separate MS-Excel spreadsheet which provides inputs to the Toolkit Software.

The Toolkit Software is written in “R” programming language, using the Jupyter Notebook interactive computational environment (version 4.0.4) with the R Kernel.

Inputs

All of the inputs to the Toolkit Software are MS Excel spreadsheet files (CSV format).

Input File Types:

There are three input file types for the Toolkit Software, described below:

A. Script

- A Script file contains each “project” to be included in the cost estimation calculations. One project per line in the script file.
- The script file naming convention is “<name>_Script.csv”. The user specifies <name> in the first line of the Toolkit Software. Example from the software: `Script_Name = "Demo_4"`
- The user needs to develop a script file for their own usage.
- There is one script file input to the Toolkit Software.
- The Columns in the script file are:
 - Input_File: project file name (See “Projects” below)
 - Mode: D – Development, M – Maintenance. Specifies if the project is in development mode or maintenance mode
 - Start_Year: 1-15. Which year to start the cost estimations. Max of 15 years.
 - Years: 1-15: Number of years for cost estimations
 - T1_Start: Trajectory 1 start year (-1 = no trajectory)
 - T1_Years: Number of years for the trajectory (-1 = no trajectory)
 - T2_Start: Trajectory 1 start year
 - T2_Years: Number of years for the trajectory
 - T3_Start: Trajectory 1 start year (Max of 3 trajectories)
 - T3_Years: Number of years for the trajectory
- An example “Demo_4_Script.csv” is provided below. This example has 3 projects, 2 of which are in maintenance mode. This example also has 2 active trajectories (T1 and T2).

Input_File	Mode	Start_Year	Years	T1_Start	T1_Years	T2_Start	T2_Years	T3_Start	T3_Years
Tank_As_Is	M	1	15	1	15	1	3	-1	-1
Tank_OA_Endstate_Dev	D	1	3	-1	-1	1	3	-1	-1
Tank_OA_Endstate_Maint	M	4	12	-1	-1	4	12	-1	-1

B. Projects

- A project file contains all of the information pertaining to a high-level software architecture needed for the toolkit software to perform life-cycle cost estimation.
- A project file can be either Maintenance mode or Development mode. For projects that need cost estimations for both development and maintenance modes, they must use different project files (via file naming conventions, see Script example above). The file formats are the same for Maintenance and Development modes.
- Project names “<project>” are found in the Input_File column of the Script file (above).
- There are two types of Project input files for each project:
 - Fixed Parameters File
 - This file contains the COCOMOII parameters that are fixed for the entire project
 - The file naming convention is “<project>_Fixed_Input.csv”
 - The columns in this file include the COCOMOII factors for this project. See COCOMOII Model Manual v2.1 for detailed explanations.
 - An example “Tank_As_Is_Fixed_Input.csv” is provided below:

OA1	OA2	OA3	OA4	FLEX	PREC	RESL	TEAM	PMAT	DATA	RUSE	DOCU	TIME	STOR	PVOL	ACAP	PCAP	PCON	APEX	PLEX	LTEX	TOOL	SITE	SCED	RELY	CPLEX	A	B	LABOR
1	1	0	-1	5.07	2.48	1.41	1.1	3.12	1.28	1	1.23	1.29	1.17	1.3	0.71	0.76	0.81	0.81	0.85	0.84	0.78	0.93	1	1	1	2.94	0.91	25000

- Component Parameters File
 - This file contains the COCOMOII parameters that apply to each software component in the project.
 - Each software component in the project is contained in a separate line in this file.
 - The 1st column is the name of the SW component.
 - The 2nd column is the estimated KSLOC for the SW component
 - The 3rd-6th columns are associated with the OA ratings for this SW component. See “OA Configurable Ratings Checklist” tool. A value of -1 indicates no OA ratings are to be accounted for regarding this component.
 - The remaining columns are the COCOMOII factors associated with the SW component. See COCOMOII Model Manual v2.1 for detailed explanations.
 - The file naming convention is “<project>_SW_Input.csv”
 - An example “Tank_As_Is_SW_Input.csv” is provided below:

Name	KSLOC	OA1	OA2	OA3	OA4	AA	DM	CM	IM	SU	UNFM	REVL	PREC	RELY	CPLX	GRWTH	SWGRW	AT
FCP-SM	150	0.77	0.95	0.4	-1	0	100	100	100	35	0.5	0	0	1	1	0.1	0.5	0
FCP-FC	250	0.77	0.95	0.4	-1	0	100	100	100	35	0.5	0	0	1	1	0.1	0.5	0
FCP-FS	100	0.77	0.95	0.4	-1	0	100	100	100	35	0.5	0	0	1	1	0.1	0.5	0
FCP-MMI	150	0.77	0.95	0.4	-1	0	100	100	100	35	0.5	0	0	1	1	0.1	0.5	0
FCP-C&C	150	0.77	0.95	0.4	-1	0	100	100	100	35	0.5	0	0	1	1	0.1	0.5	0
RT_EXEC	75	0.77	0.95	0.4	-1	0	100	100	100	35	0.5	0	0	1	1	0.1	0.5	0

C. OA-to-COCOMOII Conversion Tables

- These files are used to translate OA Ratings for each high-level software component to the applicable COCOMOII cost estimation factors within the Toolkit Software.
- There are three files of this type: “TABLE_OA1_CCII.csv”; “TABLE_OA2_CCII.csv”; and “TABLE_OA3_CCII.csv”
- These files should not be changed by the user.
- An example “TABLE_OA1_CCII.csv” is provided below:

OA1	SU_MAX	SU_MIN	UNFM_MAX	UNFM_MIN	RESL_MAX	RESL_MIN
5	25	10	0.17	0	2	1.41
4	30	15	0.33	0.17	3	2
3	35	20	0.5	0.33	4	3
2	40	30	0.67	0.5	5	4
1	45	35	0.83	0.67	6	5
0	50	40	1	0.83	7.07	6

Outputs

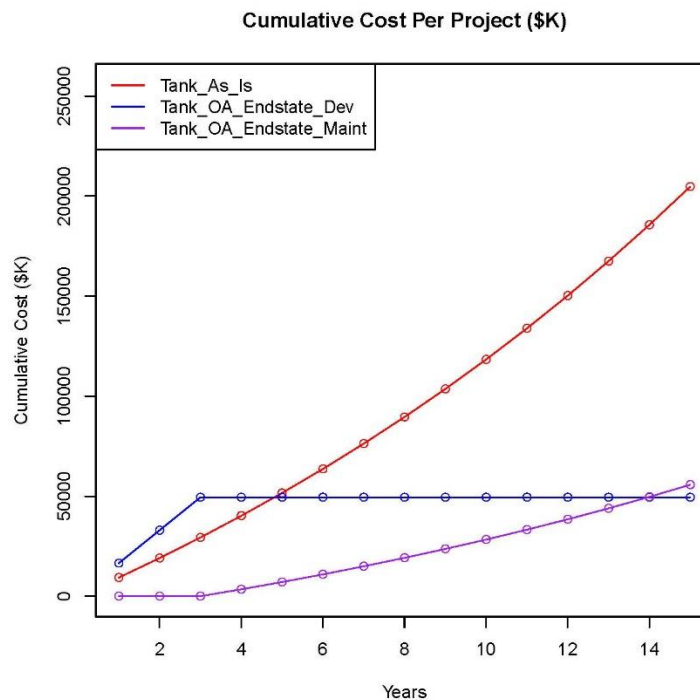
There are two types of output files for the Toolkit Software, described below:

A. Script

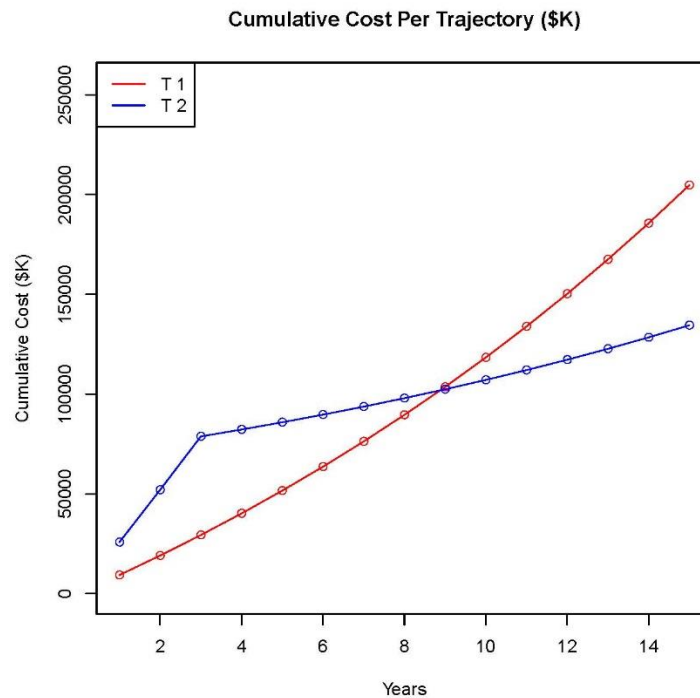
- A script file contains the yearly software cost estimates for each project in the script. Each project has two lines in a script output. The first project line is the yearly cost estimate, the second project line is the cumulative cost estimate per year. Total cumulative costs are also given.
- The script file naming convention is “<script>_Script_Output.csv”.
- There is one script file output to the Toolkit Software.
- An example “Demo_4_Script_Output.csv” is provided below:

Name	KSloc	\$K 1	\$K 2	\$K 3	\$K 4	\$K 5	\$K 6	\$K 7	\$K 8	\$K 9	\$K10	\$K11	\$K12	\$K13	\$K14	\$K15	\$K Total
Tank_As_Is	875	9280	9770	10290	10832	11404	12010	12646	13317	14018	14763	15544	16367	17232	18142	19104	204719
Tank_As_Is	875	9280	19050	29340	40172	51576	63586	76232	89549	103567	118330	133874	150241	167473	185615	204719	
Tank_OA_Endstate_Dev	685	16475	16475	16475	0	0	0	0	0	0	0	0	0	0	0	0	49425
Tank_OA_Endstate_Dev	685	16475	32950	49425	49425	49425	49425	49425	49425	49425	49425	49425	49425	49425	49425	49425	
Tank_OA_Endstate_Maint	685	0	0	0	3454	3634	3824	4026	4233	4456	4689	4934	5192	5464	5754	6054	55714
Tank_OA_Endstate_Maint	685	0	0	0	3454	7088	10912	14938	19171	23627	28316	33250	38442	43906	49660	55714	

- A script also outputs chart graphics of the cumulative yearly costs for each Project and for each Trajectory specified in the script input.
- An example “Demo_4_Script_Output” for the projects is provided below:



- An example “Demo_4_Script_Output” for the trajectories is provided below:



B. Projects

- A project file contains the yearly software cost estimates for each SW component in the project. Total cumulative costs are also given for each SW component, as well as, for each year.
- The script file naming convention is “<project>_Output.csv”.
- There is one project file output for each project in the script.
- An example “Tank_As_Is_Output.csv” is provided below:

Name	KSloc	\$K 1	\$K 2	\$K 3	\$K 4	\$K 5	\$K 6	\$K 7	\$K 8	\$K 9	\$K10	\$K11	\$K12	\$K13	\$K14	\$K15	\$K Total
FCP-SM	150	1591	1675	1764	1857	1955	2059	2168	2283	2403	2531	2665	2806	2954	3110	3275	35096
FCP-FC	250	2652	2792	2940	3095	3259	3432	3613	3805	4006	4218	4441	4676	4924	5184	5459	58496
FCP-FS	100	1060	1116	1176	1238	1303	1372	1445	1522	1602	1687	1776	1870	1969	2073	2183	23392
FCP-MMI	150	1591	1675	1764	1857	1955	2059	2168	2283	2403	2531	2665	2806	2954	3110	3275	35096
FCP-C&C	150	1591	1675	1764	1857	1955	2059	2168	2283	2403	2531	2665	2806	2954	3110	3275	35096
RT_EXEC	75	795	837	882	928	977	1029	1084	1141	1201	1265	1332	1403	1477	1555	1637	17543
Totals	875	9280	9770	10290	10832	11404	12010	12646	13317	14018	14763	15544	16367	17232	18142	19104	204719
Cumulative \$K		9280	19050	29340	40172	51576	63586	76232	89549	103567	118330	133874	150241	167473	185615	204719	

Contact Information

Mike Gagliardi

Appendix C. Security Architecture (SA) Workbook

This appendix describes the Security Architecture (SA) workbook, also called the Security Architecture Configurable Rating Checklist Tool. The workbook consists of a series of worksheets to be used in trade-off analyses when configuring and rating the costs and benefits of security architecture mechanisms. The SA workbook is based on a security analysis of scenarios developed for the quadcopter architecture and is not currently releasable. This appendix describes how the SA workbook was developed and compares it with the Open Architecture (OA) workbook, also called the Open Architecture Configurable Rating Checklist Tool, which was presented in Appendix A.

About the Quadcopter Security Architecture Workbook

This security architecture (SA) workbook contains a draft set of worksheets for evaluating a quadcopter architecture in terms of a basic set of security controls that can be implemented as architectural mechanisms. When complete, the workbook will be used in architecture trade-off analyses that consider whether an architecture includes essential security mechanisms, and calculate the cost to implement them. The set of security mechanisms to be included in an analysis will be configurable, based on a mission-focused security engineering risk assessment. In this assessment, security engineering analysts select controls and corresponding mechanisms based on the risks identified through mission thread analysis [Gagliardi 2013]. These risks are expressed in terms of threats to the mission, corresponding architectural vulnerabilities that could be exploited in an attack, and the probability and mission impact of a successful attack.

Comparison of Open Architecture (OA) Workbook with Security Architecture (SA) Workbook

The SA workbook is modeled after the Open Architecture (OA) workbook created for the US Air Force (see Appendix A). Although the two workbooks are similar in structure and appearance they differ in fundamental ways, as described below.

Maturity and Generality: The OA workbook is general and more mature.

The OA workbook can be used to evaluate the degree to which an architecture displays key attributes of open systems. The OA workbook

- Has been piloted and is relatively mature.
- Applies to architectures in general.
- Is fairly stable, due to its maturity and generality.

The SA workbook is intended to focus on the degree to which security has been designed into an architecture *for unmanned aerial vehicles (UAVs)*. However, the draft set of worksheets focus on a simple, single quadcopter architecture. In contrast with the OA workbook, the SA workbook

- Is in an initial development stage, incomplete and not ready to pilot on other architectures.
- Was developed based on an analysis of threats and vulnerabilities in operating a particular quadcopter (hobby) drone, rather than for a general UAV architecture.

- Will evolve considerably to reflect the elements of, and security risks to, more complex UAV architectures and missions, as well as results of pilot applications of the workbook.

This additional work will enable completion of the SA workbook, including the Architectural Security Mechanisms worksheet and associated cost factors; the Benefit Data worksheet, which describes the security benefits/attack types mitigated by the architectural mechanisms; and the associated Graphs worksheet.

Types of Architectural Abstractions Evaluated

In addition to differences in maturity and generality, the worksheets in the OA and SA workbooks target different architectural abstractions.

- OA workbook: Open architecture characteristics describe *structural attributes* of a system or component that confer *benefits* of openness.
- SA Workbook: Security architecture characteristics describe *mechanisms* that confer what are known in the security community as *security attributes*, i.e., the “benefit” of a security mechanism would be an improvement with respect to one or more security attributes.

Structure of the Draft SA Workbook

The structural components of the SA Workbook are described in the table below. Although the workbook is not releasable, this information conveys the intended content and use of the workbook as a tool for early-architecture security trade-off analysis.

Worksheet Name	Purpose and Description	Status
About this Workbook	Explains the origin of the workbook and the general maturity of its contents	Complete
About this tool	Provides status and a description of each of the worksheets	Complete
Arch-MechData (Security)	This is the “security checklist,” also known as the “Security Configurable Ratings Checklist.” The checklist is structured into six “security mechanism categories.” These six categories are drawn from the 20 security and privacy control families in the draft NIST 800-53v5 [NIST 2017]. Each category contains one or more architectural mechanisms (ArchMech) for implementing security controls. The Score Weight, Score, Weighted Score, and Justification for Score columns are not yet in use. The Comments and References columns provide information on SWATT implementation and recommendations found in references.	Initial conceptual draft – work remaining: Add more mechanisms (and, if applicable, more categories), determine weights, and send for broader review. Provide a means to capture and analyze cost data for implementing the security mechanisms (ultimately, this may include software, firmware, and hardware costs). Provide tailoring guidance (based on type of drone, mission risk, etc.); accordingly, enable addition/removal of mechanisms/criteria. Pilot and refine.

Worksheet Name	Purpose and Description	Status
BenefitData (Security)	The benefit data will be used to “rate” security mechanisms according to the security “benefit” conferred by the mechanism. To clearly communicate the benefit, security benefits are expressed as the traditional security attributes coupled with the attack the attribute mitigates. In an attempt to follow the OA checklist, each Security Mechanism Category is assumed to either contribute to or detract from a benefit. This may NOT hold true for security; further analysis is needed.	In work Complete: Benefits Identified Not Complete: Unweighted mapping of security mechanism categories to security benefits Determine whether it makes sense to apply weights to the mappings and (a) if so, how this should be done; (b) if not, how we should modify the approach, which was developed for open architectures, for security.
Graphs (Security)	The graphs would be used to illustrate the relationships between security mechanisms and security benefits for the architecture that is rated in the ArchMech (Security) worksheet.	Not started To generate graphs, this worksheet needs to be linked to data in completed ArchMechData (Security) and BenefitData (Security) worksheets, which are used to rate an architecture.

Guidelines for applying this SA workbook follow the guidelines developed for the OA workbook

When the SA workbook has been completed, an accompanying briefing will be developed entitled “Security Evaluation Criteria,” similar to the briefing developed for the OA workbook. As for the OA workbook, the SA workbook should NOT be used without this briefing, which will provide background, context, instructions, and examples.

The SA Workbook is not intended as a self-assessment tool for program offices. Rather, it is intended to be used by an objective, expert observer to aid in the assessment of the suitability of security mechanisms for achieving specific objectives in the context of the program.

Additional notes specific to the SA workbook

The SA workbook deals only with architecture mechanisms and is not a complete set of cybersecurity activities and mechanisms for a system or program. The latter would include activities and mechanisms in the supply chain, acquisition, risk assessment, security assessment maintenance, and other categories of security considerations.

Once completed, the SA workbook needs to undergo a series of pilots to obtain and incorporate feedback. We encourage those interested in pilot participation to contact us. We also welcome feedback on the concepts presented in this report.

Appendix D. Initial Security Analysis

The initial security analysis focused on a posited quadcopter mission thread [Gagliardi 2013]. A mission thread is a sequence of end-to-end activities and events that takes place to accomplish the execution of one or more of a system's (or system of system's) capabilities. Mission threads are developed by articulating a vignette that describes the thread, and decomposing that thread into its steps. The team applied the STRIDE model [Microsoft 2005, Howard 2006] to each step in the quadcopter thread to identify the quadcopter's susceptibility to the six means of attack and obfuscation represented by STRIDE:

- [S]poofing: Attacker posing as another entity (person or system) to gain unauthorized access to a system or capability
- [T]ampering: Unauthorized modification of data, code, or firmware at rest or in transit
- [R]epudiation: Denial by the responsible attacker that they performed a particular action or precipitated an attack
- [I]nformation Disclosure: Unauthorized exposure of information
- [D]enial of Service: Attack that denies or degrades authorized users' access to capabilities or resources
- [E]levation of privilege: Condition in which a valid user gains access to system capabilities or resources for which they are not authorized

The table below illustrates the team's use of STRIDE to analyze mission thread step 11 (S11) for the quality attribute (QA) cybersecurity. For each plausible attack on the quadcopter mission, the team established scenarios indicating the attack (stimulus), the context (environment), and how the system should respond (response). This analysis should be understood as a best-effort first result from the team rather than a final product.

QA	Mission Step	Concern (STRIDE)	Scenario <i>Stim: Stimulus, Env: Environment, and Resp: Response</i>
Cybersecurity	S11. Operator Downloads Photos from Quadcopter	General comments	General notes common to scenarios in subsequent rows (1) While all dimensions of STRIDE apply, the main purpose of attacks during this mission step is to tamper with the photo data and possibly disclose it to an unauthorized party. (2) In the Response section of each scenario, we omit the "And the system software logs information about the attack..." (or equivalent), but it should be understood that logging is included. Also, we omit the idea that to the extent the adversary or their means of access can be identified, access to the quadcopter and

QA	Mission Step	Concern (STRIDE)	Scenario <i>Stim: Stimulus, Env: Environment, and Resp: Response</i>
			ground system software may be reduced or denied, though again, it should be understood that such a notion is to be included. (3) The guiding assumption throughout the analysis of S11 is that the quadcopter has returned safely to base. If the adversary is able to physically gain access to the quadcopter or ground system software, additional attacks are possible beyond those described here.
		S: Adversary logs in as operator and impacts download of photos	1. Stim: Adversary attempts to log in as operator using stolen credentials, and thus is able to either download photos or impact the successful download of photos for the operator Env: Post-flight (after return to base) Resp: Ground operator's system software prevents adversary from using stolen credentials (e.g., by implementing multi-factor authentication)
		T: Adversary attempts to modify photos exclusive of their being downloaded	2. Stim: The activated malicious code attempts, after the quadcopter has returned to the ground, to modify any of the photos that were taken during the mission (and/or their meta-data). This type of attack includes attempting to erase storage of photo data; and perhaps attempting to re-initialize the system while photo data is in volatile memory. This scenario is largely the same as Step 5, Scenario 3. Env: Post-flight (after return to base) [Note that the environment for Step 5, Scenario 3 includes “on the ground after landing” and thus might cover the current scenario as well.] Resp: Same as Step 5, Scenario 3.
		T: Adversary attempts to modify photos during download	3. Stim: Adversary (or malicious software) attempts to change photos (and/or their meta-data) as they are moved from memory to the operator as part of the download. (Variant: the photos are downloaded but their asserted locations and number are changed; perhaps to instead point to fake photos.) This type of attack includes attempting to erase the photos; and perhaps including attempting to re-initialize the system while photo data is being transferred. Comment on Scenario: This differs from Scenario 2 as follows: Scenario 2 deals with tampering prior to (and possibly) after the

QA	Mission Step	Concern (STRIDE)	Scenario <i>Stim: Stimulus, Env: Environment, and Resp: Response</i>
			download; and does not cover tampering that occurs as part of the download operation. Env: Post-flight (after return to base) [Note that the environment for Step 5, Scenario 3 includes “on the ground after landing” and thus might cover the current scenario as well.] Resp: Quadcopter and/or Ground System software detects the tampering action; prevent the loss of data (and its inadvertent disclosure), and with only nominal delay (and interruption of service) completes the download successfully.
		R: Adversary attempts to conceal tampering or other attack	4. Stim: Adversary attempts to conceal the tampering of photo data and denies any knowledge of attack. Env: Post-flight (after return to base) Resp: Quadcopter and/or Ground System software detects the attack; records data about the attack including time, origin of attack, known effects of the attack, mitigations carried out; and secures all identify information about the source of attack that it can. The adversary is confronted and sufficient information is captured and presented to convict the adversary.
		I: Attempt to send photo data to adversary	5. Stim: Exfiltration of photos is attempted by piggy-backing on the download of the photo data—either directly, or perhaps by transferring the data to another location, e.g., on the quadcopter or ground station—for later transmittal to adversary. Comment on Scenario: This scenario is largely the same as Step 5, Scenario 5. Env: Post-flight (after return to base) Resp: The Quadcopter and/or Ground system software immediately detects that the photo data is being sent to an unauthorized location or party and prevents the transfer from occurring (e.g., by disabling the malware) while allowing the download to the authorized recipient to proceed.
		D: Attempt to disrupt successful download	6. Stim: Adversary (e.g., through ground system or quadcopter malware) attempts to flood the quadcopter and/or ground station assets (camera, storage, communication channels, PX4 Commander software) with requests (e.g., re-initialization) that disrupt the downloading of photos.

QA	Mission Step	Concern (STRIDE)	Scenario <i>Stim: Stimulus, Env: Environment, and Resp: Response</i>
		of photo data	Comment on Scenario: This scenario is largely the same as Step 5, Scenario 6, but stated in more general terms. Env: Post-flight (after return to base) Resp: The Quadcopter and/or Ground station software immediately detects the attempt to flood the system with requests, disables malware, and resumes downloading of photo data with minimal interruption
		E: Adversary exploits vulnerability to improve access to photos	7. Stim: Adversary exploits a vulnerability in the ground system and/or quadcopter malware to gain elevated privileges with which to embark on further mischief, including one of the previous scenarios. (The attack may proceed by re-initializing the software and gaining root control of the quadcopter.) Env: Post-flight (after return to base) Resp: The Quadcopter and/or Ground station software detects the attempt at elevating privilege and denies the privileges from being granted to the adversary, logs the attempt (including for further remediation to remove the vulnerability), and denies the adversary further access to the quadcopter or ground system software.

Following the analysis, the next step is to identify probability and mission impact of each successful attack scenario and to propose security controls and corresponding architectural mechanisms to resist and recover from the attacks. These would be captured in the SA workbook for further analysis and rating.

A comprehensive trade-off analysis would involve more variables, including identifying alternatives for the mechanisms, estimating cost and schedule to implement each, and determining effects on other attributes such as reliability and maintainability, and on factors such as performance and capability.

Appendix E. Quadcopter architecture as revealed by code

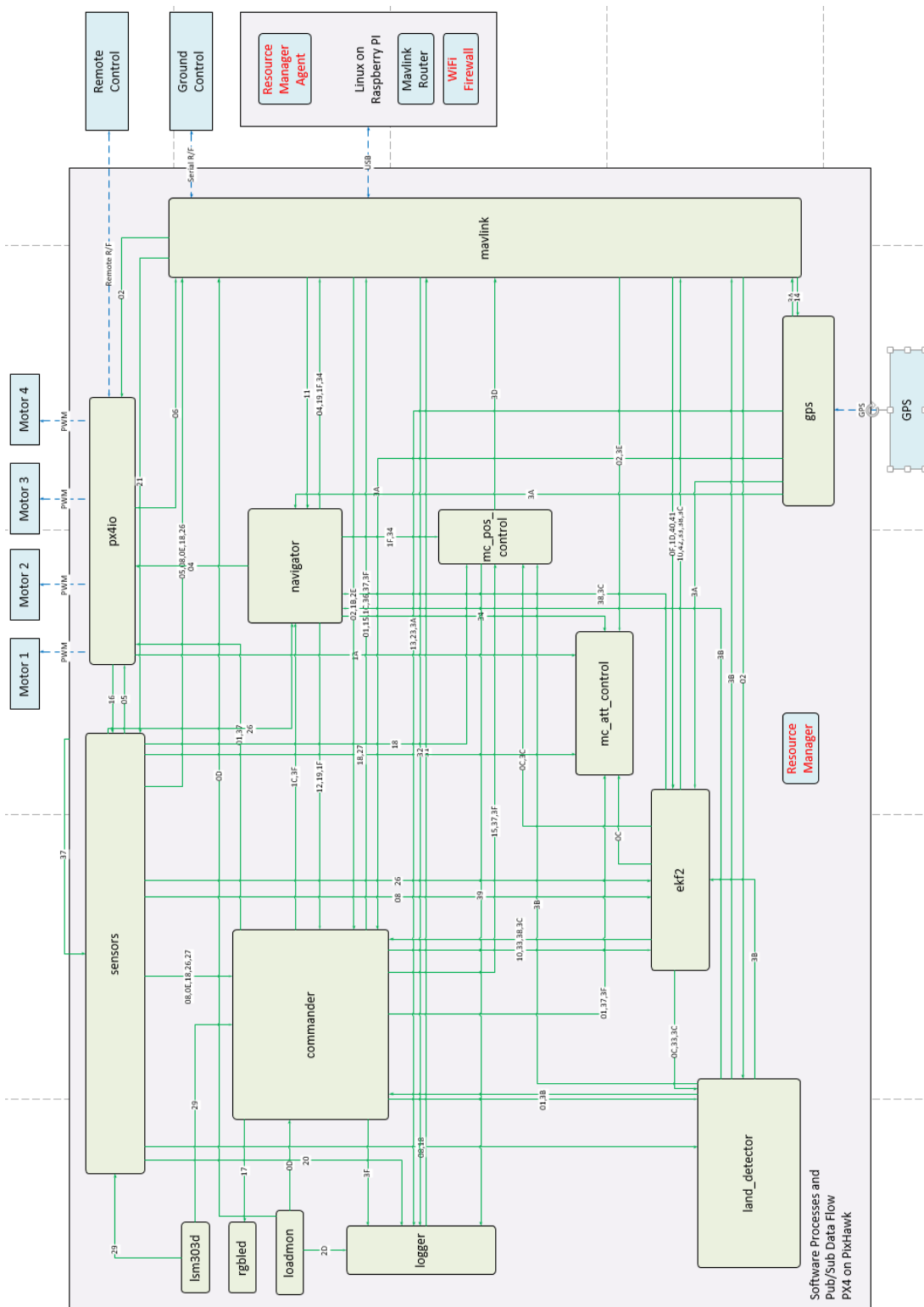


Figure G-1. Quadcopter Software Architecture Reverse Engineered from Code

References

URLs are valid as of the publication date of this document.

[Boehm 2000]

Boehm, B; Abts, C; Brown, A; Chulani, S; Clark, B; Horowitz, E; Madachy, R; Reifer, D; & Steece, B. *Software Cost Estimation with COCOMO II*. Englewood Cliffs, NJ: Prentice-Hall, 2000. ISBN 0-13-026692-2.

[CNSS 2013]

Committee on National Security Systems. *CNSSI 1253: Space Platform Overlay*. June 2013. <https://www.cnss.gov/CNSS/issuances/Instructions.cfm>

[CNSS 2014]

Committee on National Security Systems. *CNSSI 1253F Attachment 2: Space Overlay*. March 2014. <https://www.cnss.gov/CNSS/issuances/Instructions.cfm>

[DJI 2016]

DJI. *Flame Wheel ARF KIT*. Flame wheel F450 frame. Retrieved December 2016. <https://www.dji.com/flame-wheel-arf>

[Dronecode 2016]

Dronecode Project. *PX4: The Professional Autopilot*. Px4 open-source autopilot software. Retrieved December 2016. <http://px4.io/>

[Gagliardi 2013]

Gagliardi, M.; Wood, B.; & Morrow, T. *Introduction to the Mission Thread Workshop*. CMU/SEI-2013-TR-003. Software Engineering Institute, Carnegie Mellon University. 2013. <https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=63148>

[Gagliardi 2016]

Gagliardi M. & Konrad, M. *SEI SWATT FY16 LENS Toolkit Overview and Demonstration*. PowerPoint presentation, 5/16/2016.

[Holland 2016]

Holland J. *Engineered Resilient Systems*. NDIA Systems Engineering Conference, PowerPoint briefing, October 2015.

[Howard 2006]

Howard, Michael & Lipner, Steve. *The Security Development Life Cycle*. Microsoft Press, 2006.

[Meier 2016]

Meier, Lorenz. *What is Pixhawk*, Pixhawk flight management unit. Retrieved December 2016. <https://pixhawk.org/>.

[Microsoft 2005]

Microsoft Corporation. *The STRIDE Threat Model*. 2005
[https://msdn.microsoft.com/en-us/library/ee823878\(v=cs.20\).aspx](https://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx)

[NIST 2014a]

National Institute for Standards and Technology. *Framework for Improving Critical Infrastructure Cybersecurity*, Version 1.0. February 2014.
<https://www.nist.gov/document-3766>

[NIST 2014b]

National Institute for Standards and Technology. *Security and Privacy Controls for Information Systems and Organizations*. SP 800-53 Rev. 4. February 2014.
<https://csrc.nist.gov/publications/detail/white-paper/2014/02/19/summary-of-nist-sp-800-53-rev-4-security--privacy-controls/final>

[NIST 2017a]

National Institute for Standards and Technology. *Framework for Improving Critical Infrastructure Cybersecurity*, Version 1.1. Draft 2, December 2017.
<https://www.nist.gov/cybersecurity-framework/cybersecurity-framework-draft-version-11>

[NIST 2017b]

National Institute for Standards and Technology. *Security and Privacy Controls for Information Systems and Organizations*. SP 800-53 Rev. 5 (draft). August 2017.
<https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/draft>

[Sheard 2015]

Sheard, S; Shull, F; Bachmann, F; & Creel, R. *Software Attributes Trade-off Tool (SWATT)*. FY2016 LENS proposal, 7/20/2015.

[Sheard 2016]

Sheard, S. & Shull, F. *Software Attributes Trade-off Tool (SWATT)*. FY2017 Line proposal, 3/4/2016.

[Shull 2015]

Shull, F. & Sheard, S. *FY2016 LENS Projects Software Attributes Trade-off Tool (SWATT)*. Power-Point presentation, 8/27/2015.

[USC 2000]

University of Southern California. *COCOMO II Model Definition Manual, version 2.1*. USC Center for Software Engineering, 2000.
http://sunset.usc.edu/cse/pub/research/COCOMOII/cocomo_main.html

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave Blank)		2. REPORT DATE September 2018		3. REPORT TYPE AND DATES COVERED Final
4. TITLE AND SUBTITLE Modeling the Effects of Software-Related Decisions on Early System Cost Estimates: Experience Report from the Software Attributes Trade-off Tool (SWATT) Project			5. FUNDING NUMBERS FA8721-05-C-0003	
6. AUTHOR(S) Sarah Sheard, Rita Creel, Patrick Donohoe, Michael J. Gagliardi, Michael D. Konrad, Gabriel A. Moreno				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213			8. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-2018-SR-029	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFLCMC/PZE/Hanscom Enterprise Acquisition Division 20 Schilling Circle Building 1305 Hanscom AFB, MA 01731-2116			10. SPONSORING/MONITORING AGENCY REPORT NUMBER n/a	
11. SUPPLEMENTARY NOTES cybersecurity, software architecture, software development, software cost, software maintenance, software performance				
12A DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS			12B DISTRIBUTION CODE	
13. ABSTRACT (MAXIMUM 200 WORDS) This report describes the results obtained and the experiences gained on a two-year (FY2016-2017) SEI project named the Software Attributes Trade-off Tool (SWATT) project that investigated the relationship of software design decisions to key system-level performance parameters and costs. The FY2016 project created a tool that takes the results of a software openness checklist and converts it into an estimate of software development and maintenance costs. The FY2017 project evaluated how to address the security issues that mission sequences create, when the scenarios to be used depend on architectural features. With significant analyst assistance, this tool can be used first to determine what security concerns a given software architecture may have and then to estimate development and maintenance costs for a number of architectural options that improve security countermeasures.				
14. SUBJECT TERMS			15. NUMBER OF PAGES 73	
16. PRICE CODE				
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. Z39-18
298-102