PROCESS

AGILE & ITERATIVE DEVELOPMENT (CHAPTER 1)

A MANAGER'S GUIDE BY: CRAIG LARMAN

WEEK 1: INTRODUCTION - DEVELOPMENT PROCESSES, ISO 12207 AND CMMI

BY: JOSEPH MARTINAZZI

UBIQUITY OF COMPUTING

TODAY SOFTWARE HAS BECOME EMBEDDED INTO TECHNOLOGY INDIVIDUALS USE AS A PART OF THEIR DAILY LIVES.

THIS INCLUDES APPLICATIONS ON A COMPUTER THAT AN INDIVIDUAL CAN CHOOSE TO USE:

- USING WORD PROCESSING/SPREAD SHEET APPLICATIONS ON A COMPUTER,
- USING THE INTERNET FOR RESEARCH AND E-COMMERCE, AND
- PLAYING COMPUTER GAMES.

AS WELL AS EMBEDDED SOFTWARE IN MOST DEVICES THAT INDIVIDUALS DON'T HAVE A CHOICE TO USE:

- USING A SMART PHONE OR MOST OTHER CONSUMER ELECTRONIC DEVICES,
- DRIVING A CAR
- FLYING ON AN AIRPLANE,
- USING ELECTRICITY OR DRINKING WATER, AND
- USING MEDICAL EQUIPMENT TO DIAGNOSE/TREAT ILLNESSES.

UBIQUITY OF COMPUTING

AS A RESULT, THERE IS A LOT OF **RESPONSIBILITY** ON **COMPANIES AND INDIVIDUALS** THAT WORK AT THESE COMPANIES TO TAKE CONSUMER PRIVACY & SAFETY INTO ACCOUNT WHEN DEVELOPING THESE PRODUCTS.

A WAY FOR A COMPANY TO ACHIEVE THIS GOAL IS TO ESTABLISH A WELL-DEFINED PROCESS AND TO ENSURE ITS EMPLOYEES FOLLOW THAT PROCESS!

UNFORTUNATELY, SOFTWARE PROGRAMS HAVE BECOME MORE AND MORE COMPLEX AND THEIR USE MAY NOT ALWAYS BE POSSIBLE TO PREDICT.

- SOFTWARE IS DEVELOPED BY PEOPLE WHO THINK SEQUENTIALLY, WHO HAVE LIMITED DOMAIN KNOWLEDGE, AND WHO MAKE MISTAKES
- IMPOSSIBLE TO DETERMINE THE EFFECTS OF CONCURRENCY (MULTIPLE PROCESSES AND EXTERNAL INPUTS)
- INABILITY TO VERIFY & VALIDATE SYSTEMS IN AN ACCURATE TEST ENVIRONMENT
- ACCEPTANCE OF FAULT TOLERANT SYSTEMS

UBIQUITY OF COMPUTING – IN SMART PHONES

EXAMPLE #1: **SMART PHONES AND THE IMPACT THEY HAVE ON SOCIETY?**

- SMARTPHONES HAVE ENABLED MANY NEW WAYS FOR PEOPLE TO CONNECT WITH ONE ANOTHER OUTSIDE OF CONVERSATION INCLUDING:
    - FACETIMING,
    - TEXTING,
    - TAKING AND SHARING PICTURES,
    - ACCESSING EMAIL (OVER THE INTERNET), AND
    - ACCESSING SOCIAL MEDIA SITES (OVER THE INTERNET).

- SMARTPHONE APPS HAVE ENDLESS USES THAT ENABLE INDIVIDUALS TO STREAM VIDEOS, LISTEN TO MUSIC, AND EVEN GET A FAST PASS TICKET AT DISNEYLAND.

UBIQUITY OF COMPUTING – IN SMART PHONES

EXAMPLE #1: **SMART PHONES AND THE IMPACT THEY HAVE ON SOCIETY?**

- UNFORTUNATELY, SMARTPHONES HAVE CREATED MANY UNFORESEEN ISSUES IN SOCIETY AS WELL:
  - USE OF SMARTPHONES WHILE DRIVING INCREASE THE RISK OF AN ACCIDENT.
  - PEOPLE USE SMARTPHONES IN INAPPROPRIATE PLACES AND THE FACT THAT THEY HAVE CAMERAS AFFECTS OUR PRIVACY IN PUBLIC AND NON-PUBLIC PLACES.
  - RESEARCHERS ARE LEARNING AN ENORMOUS AMOUNT ABOUT OUR BEHAVIOR FROM HOW WE USE OUR SMARTPHONE. INVASION OF PRIVACY

UBIQUITY OF COMPUTING – IN CRITICAL SYSTEMS

EXAMPLE #2: **POORLY DESIGNED USER INTERFACES IN SAFETY CRITICAL SYSTEMS**

CASE STUDY #1 - ISSUES RESULTING FROM POORLY DESIGNED USER INTERFACES RESULTED IN THE CRASH OF AMERICAN AIRLINES FLIGHT 965 NEAR CALI, COLOMBIA.

- THE PILOT INTENDED TO LOCK THE AUTOPILOT ONTO A BEACON WHILE APPROACHING THE AIRPORT. AFTER ENTERING “R”, THE COMPUTER SYSTEM DISPLAYED 6 BEACONS WITH “R”. NORMALLY, THE CLOSET BEACON IS DISPLAYED AT THE TOP OF THE LIST.
- IN THIS CASE THE BEACON AT THE TOP OF THE LIST WAS 100 MILES AWAY RESULTING IN THE PLANE TURNING MORE THAN 90 DEGREES CRASHING INTO A MOUNTAIN, ALL 159 PEOPLE ON BOARD WERE KILLED.

WHY WAS THIS INCONSISTENCY NOT DISCOVERED DURING TESTING?

WAS THERE SOME LATEN ERROR THAT ONLY OCCURRED BASED ON A CERTAIN SET OF CIRCUMSTANCES OR WAS THERE A PROCESS PROBLEM?

UBIQUITY OF COMPUTING – IN CRITICAL SYSTEMS

EXAMPLE #2: **POORLY DESIGNED USER INTERFACES IN SAFETY CRITICAL SYSTEMS**

CASE STUDY #2 - ISSUES RESULTING FROM POORLY DESIGNED USER INTERFACES RESULTED IN ASIANA AIRLINES FLIGHT 214 CRASHING IN SAN FRANCISCO.

- THE PILOT DID NOT REALIZE THAT THE SPECIFIC AUTOPILOT MODE HE SELECTED DISENGAGED AN AUTO-THROTTLE FEATURE RESULTING IN THE PLANE'S SPEED DECREASING TOO RAPIDLY ON APPROACH TO THE AIRPORT.
- IN THIS CASE THE TAIL OF THE PLANE BROKE OFF KILLING 3 PASSENGERS AND INJURING THE REST OF THE PASSENGERS.

WHY WAS THE PILOT UNAWARE OF THIS FEATURE IN THIS AIRCRAFT?

WAS THERE AN ISSUE WITH THE TRAINING MATERIAL OR ASSOCIATED FLIGHT SIMULATOR?

UBIQUITY OF COMPUTING – IN CRITICAL SYSTEMS

EXAMPLE #3: **FAILURE TO CORRECTLY ESTABLISH SAFETY AS A KEY PART OF A PROCESS.**

PEOPLE MAKE DECISIONS BASED ON FACTS BUT TEND TO ERROR ON THE SIDE OF CAUTION IN ABSENCE OF A CONVINCING CASE FOR SAFETY.

CASE STUDY #1 - **THE SPACE SHUTTLE CHALLENGER WAS DESTROYED AS A RESULT OF A BLOW BY** (BREACH IN RUBBER GASKET THAT ENABLED BURNING GAS TO IGNITE THE ROCKET FUEL).

- NASA HAD ORIGINALLY HALTED SHUTTLE OPERATIONS UNTIL THE BLOW BY ISSUE COULD BE RESOLVED. HOWEVER, ONCE A SOLUTION WAS IDENTIFIED AND WENT INTO PRODUCTION, SHUTTLE LAUNCHES WERE PERMITTED TO CONTINUE. FAILURE TO MAKE A CONVINCING CASE FOR SAFETY
- THE NIGHT BEFORE THE SHUTTLE LAUNCH WAS THE COLDEST NIGHT ON RECORD (RUBBER GETS BRITTLE WHEN IT IS COLD). ENGINEERS ARGUED FOR A DELAY BECAUSE THEY KNEW THE COLD WEATHER POSED A SEVERE THREAT. HOWEVER, IN THE END THE LAUNCH WENT AHEAD AS SCHEDULED SINCE THEY COULD NOT ABSOLUTELY PROVE THAT THE SYSTEM WAS NOT SAFE UNDER THE CURRENT CONDITIONS. FAILURE TO MAKE A CONVINCING CASE FOR SAFETY
- THE SHUTTLE EXPLODED ON TAKEOFF KILLING A SCHOOLTEACHER, A SCIENTIST FROM HUGHES AIRCRAFT COMPANY, AND ALL THE ASTRONAUTS ON BOARD.

UBIQUITY OF COMPUTING – IN CRITICAL SYSTEMS

EXAMPLE #3: **FAILURE TO CORRECTLY ESTABLISH SAFETY AS A KEY PART OF A PROCESS.**

PEOPLE MAKE DECISIONS BASED ON FACTS BUT TEND TO ERROR ON THE SIDE OF CAUTION IN ABSENCE OF A CONVINCING CASE FOR SAFETY.

CASE STUDY #2 – **THE CHERNOBYL NUCLEAR DISASTER**

- WAS A RESULT OF THE PLANT OPERATORS NOT UNDERSTANDING THE RAMIFICATIONS OF HAVING THE PLANT ONLINE FOR TWO YEARS EVEN THOUGH THE THEY KNEW THE BACKUP SYSTEMS COULD NOT OPERATE FOR 60-75 SECONDS IN THE EVENT OF AN ELECTRICAL POWER FAILURE.
- THESE OPERATORS FELL UNDER THE TRAP OF THINKING IT WAS OK TO CONTINUE OPERATIONS BECAUSE THEY WERE ATTEMPTING TO SOLVE THE PROBLEM.

UBIQUITY OF COMPUTING – IN CRITICAL SYSTEMS

EXAMPLE #3: **FAILURE TO CORRECTLY ESTABLISH SAFETY AS A KEY PART OF A PROCESS.**

> PEOPLE MAKE DECISIONS BASED ON FACTS BUT TEND TO ERROR ON THE SIDE OF CAUTION IN ABSENCE OF A CONVINCING CASE FOR SAFETY.

CASE STUDY #3 – **THE CRASHING OF 2 BOEING 800-MAX JET AIRLINERS**.

UNDERSTANDING THE PROBLEM YOU ARE TRYING TO SOLVE USING SOFTWARE IS EXTREMELY IMPORTANT PRIOR TO DESIGNING THE SOLUTION.

- THE DESIGNERS CHOSE TO REUSE AN EXISTING HARDWARE DESIGN VS. CREATING A NEW DESIGN TO SAVE TIME AND MONEY.
- AT SOME POINT IN THE DEVELOPMENT PROCESS, THE ENGINEERS MUST HAVE REALIZED THE PLANE HAD AN ISSUE DUE TO WEIGHT DISTRIBUTION AND DECIDED TO USE SOFTWARE TO CORRECT THE ISSUE OF THE NOSE OF PLANE POINTING DOWN DURING TAKE-OFF.

> *DO YOU THINK USING SOFTWARE TO CORRECT A HARDWARE ISSUE WITH THE PLANE WAS THE RIGHT WAY TO GO?*
>
> *DO YOU THINK THEIR PROCESS HAD ADEQUATE CHECKS FOR SAFETY?*

UBIQUITY OF COMPUTING – FUTURE TECHNOLOGY

EXAMPLE #4: **SELF-DRIVING CARS** – ARE THEY GOOD OR BAD FOR SOCIETY?

UNDERSTANDING THE PROBLEM YOU ARE TRYING TO SOLVE USING SOFTWARE IS EXTREMELY IMPORTANT PRIOR TO DESIGNING THE SOLUTION.

- WILL THEY SAVE MONEY, OR WILL ROAD SYSTEMS NEED TO EQUIP WITH ADDITIONAL SENSORS TO AID THE FULLY AUTOMATED VEHICLES?
- WILL THEY REDUCE TRAFFIC (BY PICKING UP MULTIPLE INDIVIDUALS ON THEIR WAY TO WORK) OR WILL THEY CAUSE MORE CONGESTION (EMPTY VEHICLES PICK UP 1 INDIVIDUAL)?
- WILL THEY SAVE LIVES (95% OF ACCIDENTS ARE CAUSED BY HUMAN ERROR) OR WILL THEY PUT THE OCCUPANTS AT RISK (COMPUTERS CAN BE HACKED).

IN CASES WHERE A CRASH IS UNAVOIDABLE; HOW WILL THE SOFTWARE DECIDE WHAT OR WHO TO HIT?

SHOULD THE SOFTWARE ALWAYS PRIORITIZE THE LIVES OF INDIVIDUALS WITHIN THE VEHICLE OR SHOULD ITS CRASH AVOIDANCE ALGORITHM BE BASED ON SAVING THE GREATEST NUMBER OF LIVES?

IMPORTANCE OF ORGANIZATIONAL PROCESSES

WHAT IS A DEVELOPMENT PROCESS?

- A DEVELOPMENT PROCESS OR PROCESSES ARE USED TO DEFINE A SYSTEMATIC, DISCIPLINED, AND QUANTIFIABLE APPROACH TO THE DEVELOPMENT, OPERATION, AND MAINTENANCE OF AN END-PRODUCT.

WHY IS FOLLOWING A PROCESS IMPORTANT?

- PROCESS LAYS THE FOUNDATION OF HOW AN ORGANIZATION DEVELOPS WORK PRODUCTS, ESTABLISHES MILESTONES, ENSURES QUALITY, ENSURES SAFETY (WHEN APPLICABLE) AND MANAGES CHANGE TO ENSURE THE END-PRODUCT THEY PRODUCE MEETS THE NEEDS OF THEIR CUSTOMER IN A TIMELY MANNER.
- PROCESSES ARE USED IN THE PLANNING, SPECIFICATION, DESIGN, IMPLEMENTATION, AND TEST OF SPECIFICATIONS, SOFTWARE, AND HARDWARE TO SUPPORT INTEGRATED PRODUCT TEAMS AND PROGRAMS GOALS.
- IN ADDITION, PROCESSES ARE USED TO MANAGE VARIOUS ASPECTS OF PROGRAMS INCLUDING PROJECT TRACKING (COST & SCHEDULE), RISK MANAGEMENT, WORK PRODUCT PREPARATION AND PRODUCTION, PRODUCT REUSABILITY AND SIZE MEASUREMENT, CONFIGURATION MANAGEMENT, QUALITY ASSURANCE, AND TECHNICAL REVIEW PERIODICITY AND CONTENT.
- HOW WELL AN ORGANIZATION FOLLOWS IT PROCESSES ACTUAL IMPACTS ITS ABILITY TO BID ON GOVERNMENT RELATED CONTRACTS.

CAPABILITY MATURITY MODEL INTEGRATION (CMMI)

- CMMI IS A PROCESS MODEL DEVELOPED BY THE SOFTWARE ENGINEERING INSTITUTE AT CARNEGIE MELLON UNIVERSITY AS MECHANISM TO IMPROVE PERFORMANCE THROUGHOUT AN ORGANIZATION.

- THE GOAL OF THE MODEL IS FOR ORGANIZATIONS TO CREATE A SET OF BEST PRACTICES FOR RESOLVING PROCESS ISSUES, MINIMIZING PROGRAM RISKS, AND CREATING A QUALITY PRODUCT IN THE MOST EFFICIENT MANNER.

- ORGANIZATIONS THAT HAVE MASTERED THESE PRACTICES CAN BE ASSESSED BY THE CMMI INSTITUTE TO DETERMINE WHAT CMMI MATURITY LEVEL THEY ARE OPERATING AT.

- THE GOVERNMENT TYPICALLY REQUIRES AN ORGANIZATION TO HAVE A CMMI MATURITY LEVEL OF ANYWHERE FROM A 3 TO A 5 TO EVEN BE CONSIDERED AS A CANDIDATE TO BID OR WORK ON A US GOVERNMENT CONTRACT. ORGANIZATIONS THAT OBTAIN A CMMI MATURITY LEVEL OF 4 OR 5 ARE VIEWED TO BE MATURE.

CAPABILITY MATURITY MODEL INTEGRATION (CMMI)

- **STANDARD CMMI APPRAISAL METHOD FOR PROCESS IMPROVEMENT (SCAMPI)** - IS THE OFFICIAL METHOD USED BY THE CMMI INSTITUTE TO EVALUATE AN ORGANIZATIONS LEVEL OF MATURITY. THERE ARE THREE APPRAISAL CLASSES: CLASS A, CLASS B, AND CLASS C.
  - CLASS A – SCAMPI A IS THE MOST COMPREHENSIVE OF THE APPRAISAL CLASSES AND RESULTS IN PROVIDING THE ORGANIZATION WITH AN OFFICIAL MATURITY LEVEL RATING.
  - CLASS B – SCAMPI B IS LESS A RIGOROUS APPRAISAL METHOD. THIS APPRAISAL IS USEFUL WHEN AN ORGANIZATION WANTS TO PERFORM AN INTERNAL SELF APPRAISAL TO DETERMINE WHAT MATURITY LEVEL THEY WOULD ACHIEVE IF A SCAMPI A AUDIT WAS CONDUCTED. THIS IS USED TO FIND POTENTIAL ISSUES AND CORRECT PROBLEMS PRIOR TO GOING THROUGH THE OFFICIAL SCAMPI A ASSESSMENT.
  - CLASS C – SCAMPI C IS A SHORT FLEXIBLE APPRAISAL METHOD THAT ASSISTS AN ORGANIZATION’S BEST PRACTICES AND HOW WELL THEY ALIGN WITH CMMI PRACTICES. IT CAN BE USED AT A HIGH-LEVEL TO ADDRESS ORGANIZATIONAL ISSUES OR AT A LOWER-LEVEL TO ADDRESS PROGRAM OR PROCESS ISSUES AND TO ADDRESS SPECIFIC RISK AREAS.

CAPABILITY MATURITY MODEL INTEGRATION (CMMI)

CMMI MATURITY LEVELS

- **CMMI LEVEL 1: INITIAL** – PROCESSES WITHIN THE ORGANIZATION ARE NOT WELL DEFINED AND MAY NOT BE REPEATABLE. THE ORGANIZATION RELIES ON KEY INDIVIDUALS TO KEEP THINGS RUNNING AND IS MORE REACTIVE VS. PROACTIVE IN MANAGING PROJECTS. PROGRAMS TYPICALLY DO NOT GET COMPLETED WITHIN COST OR SCHEDULE DUE TO INEFFICIENCIES.
- **CMMI LEVEL 2: MANAGED AND REPEATABLE** – PROCESSES WITHIN THE ORGANIZATION ARE DEFINED AND PRODUCE REPEATABLE RESULTS. THE ORGANIZATION HAS ACHIEVED A BASIC LEVEL OF PROJECT MANAGEMENT IN WHICH PROGRAMS ARE PLANNED, REQUIREMENTS MANAGED, AND PROCESSES/WORK PRODUCTS ARE MONITORED, MEASURED, AND CONTROLLED.
- **CMMI LEVEL 3: DEFINED** – PROCESSES WITHIN THE ORGANIZATION ARE STANDARDIZED TO PROVIDE CONSISTENT RESULTS ACROSS PROGRAM EXECUTION. KEY PROGRAM AND TECHNICAL PROCESSES INCLUDE INTEGRATED PROGRAM MANAGEMENT, CONFIGURATION MANAGEMENT, REQUIREMENTS DEVELOPMENT, RISK MANAGEMENT, CAUSAL ANALYSIS AND RESOLUTION, DESIGN, TEST, INTEGRATION, VERIFICATION & VALIDATION AND TRAINING.
- **CMMI LEVEL 4: QUANTITATIVELY MANAGED** – PROCESSES WITHIN THE ORGANIZATION ARE MATURE ENOUGH THAT THEY CAN BE MEASURED USING DEFINED METRICS TO MINIMIZE PROGRAM RISKS AND CORRECT PROCESS DEFICIENCIES.
- **CMMI LEVEL 5: OPTIMIZING** – PROCESSES WITHIN THE ORGANIZATION ARE MATURE ENOUGH THAT THEY ARE BOTH STABLE AND FLEXIBLE ALLOWING FOR CONTINUOUS PROCESS IMPROVEMENT AS NEW TECHNOLOGY IS INCORPORATED INTO THEIR WORK PRODUCTS.

KEY MANAGEMENT AND DISCIPLINE PROCESSES

Integrated Program Management

Program Organization

Key Program Processes: 1. Program Schedule and Milestones, 2. Program Work Breakdown Structure/Cost Collection Method, 3. Program Requirements, 4. Program Verification & Validation Plan, 5. Program Change Management Plan, 6. Program Risk Management Plan, and 7. Program Stake Holder Involvement Plan, 8. Program Quality Assurance Plan

Systems & Hardware Engineering Disciplines

Systems Organization

Key Systems Processes: 1. System Requirement Specifications, 2. Software Requirements Specifications (SRS), 3. Systems Verification and Validation Plan.

Hardware Organization

Software Engineering Discipline

Software Organization

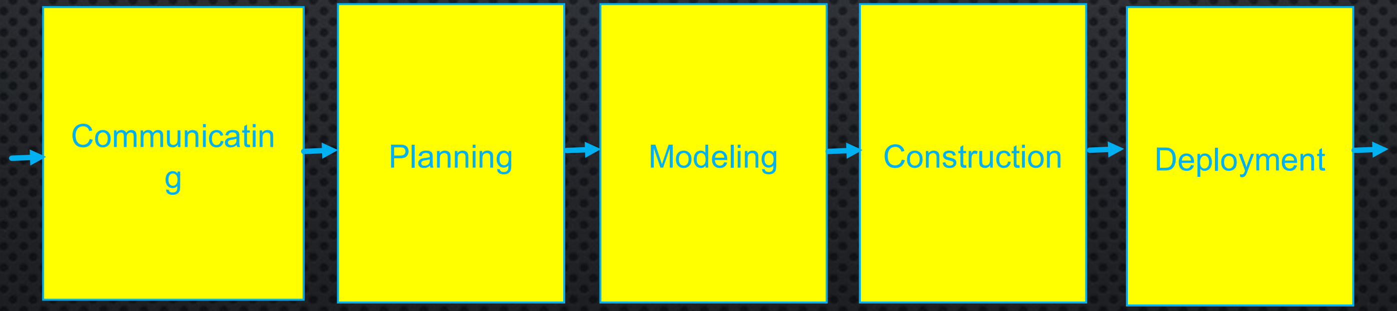
Key Software Processes: 1. Software Development Plan (SDP), 2. Software Build Plan (SBP), 3. Software Preliminary and Detail Design, 4. Software Code & Unit Test Plan, 5. Software Coding Standards (Language Specific), 6. Software Integration Plan, 7. Software Verification Plan, 8. Software Configuration Management Plan

PROCESS FLOW

WHAT IS A PROCESS FLOW?

- A PROCESS TYPICALLY OCCURS WITHIN A GENERIC PROCESS FRAMEWORK OF COMMUNICATING, PLANNING, MODELING, CONSTRUCTION, AND DEPLOYMENT.
- A PROCESS FLOW CAN BE LINEAR, ITERATIVE, EVOLUTIONARY OR PARALLEL.

LINEAR PROCESS FLOW



Communicating

Planning

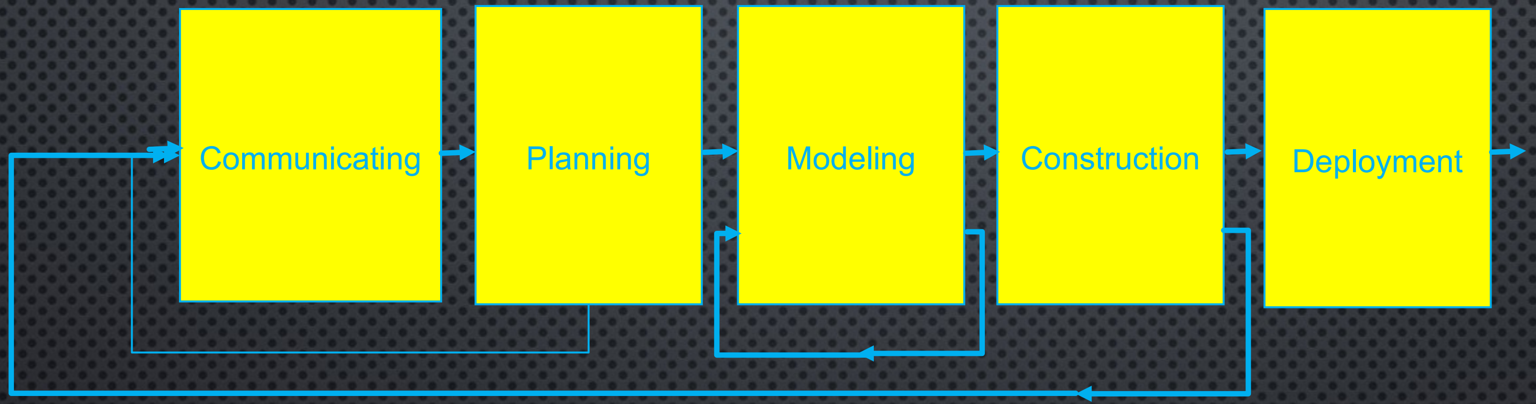
Modeling

Construction

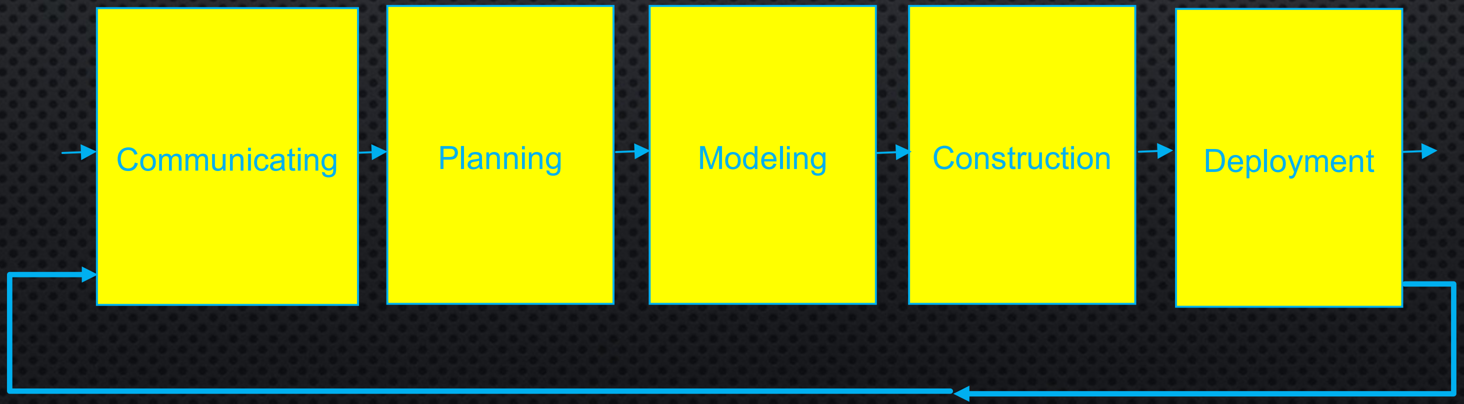
Deployment

PROCESS FLOW

ITERATIVE PROCESS FLOW

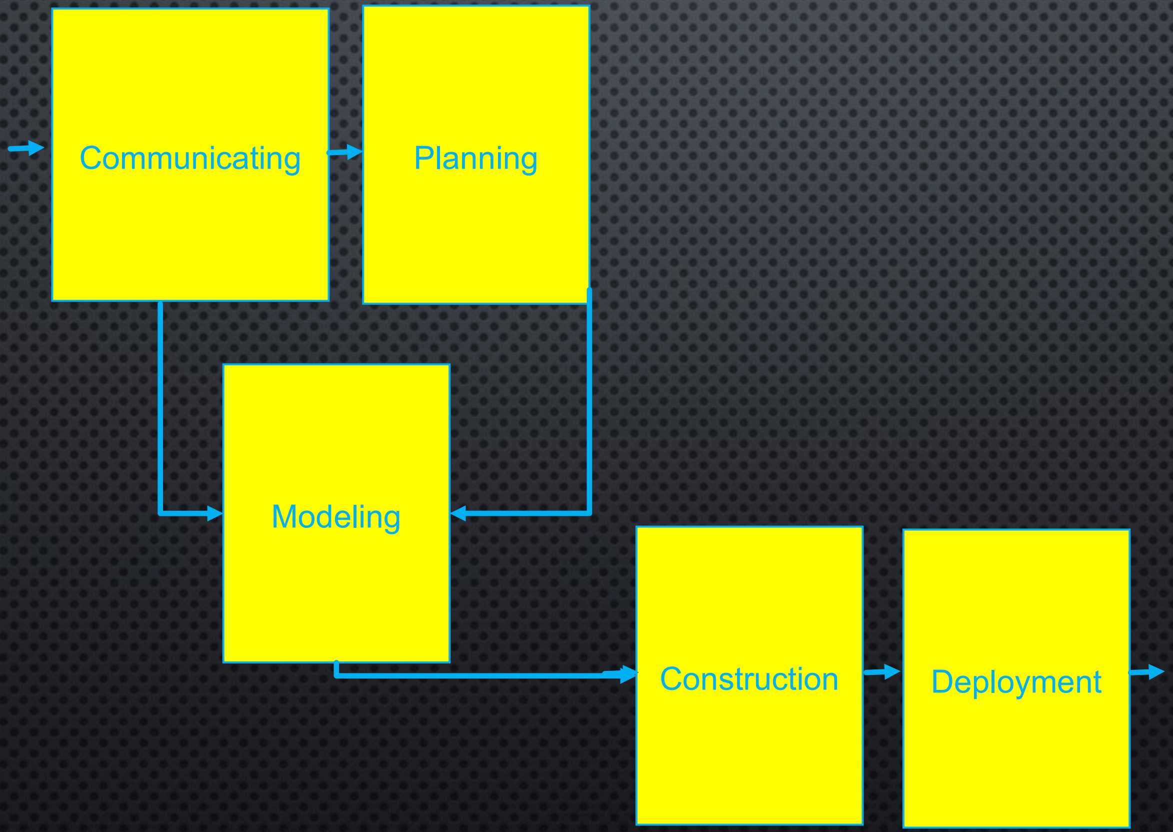


EVOLUTIONARY PROCESS FLOW



PROCESS FLOW

PARALLEL PROCESS FLOW



PROCESS MODEL

WHAT IS A PROCESS MODEL?

- A PROCESS MODEL IS USED TO DEFINE A SYSTEMATIC, DISCIPLINED, AND QUANTIFIABLE APPROACH TO THE DEVELOPMENT, OPERATION, AND MAINTENANCE OF A SOFTWARE WORK PRODUCT.
- PROCESS MODELS CAN BE PRESCRIPTIVE (IN WHICH TASKS ARE COMPLETED IN A SEQUENTIAL FASHION) OR INCREMENTAL (IN WHICH TASKS ARE COMPLETED IN LINEAR & PARALLEL FASHION) AND EVOLUTIONARY (IN WHICH TASKS ARE COMPLETED INCREMENTALLY WITH EACH INCREMENT PROVIDING MORE CAPABILITY).

PRESCRIPTIVE PROCESS MODELS

- WATERFALL LIFECYCLE MODEL – IS A SEQUENTIAL DEVELOPMENT MODEL. IT WAS THE PRIMARY MODEL USED IN DEPARTMENT OF DEFENSE (DOD) CONTRACTS FROM THE 1980’S-1990’S. UNFORTUNATELY, MANY OF THESE PROGRAMS FAILED TO PRODUCE THE END PRODUCT WITHIN COST AND SCHEDULE AND MANY FAILED TO PRODUCE AN END PRODUCT AT ALL.
- VERIFICATION & VALIDATION (V) MODEL – IS A SEQUENTIAL DEVELOPMENT MODEL.

PROCESS MODEL

ITERATIVE AND EVOLUTIONARY PROCESS MODELS

- **PROTOTYPING MODEL** – ALTHOUGH PROTOTYPING CAN BE USED AS A STAND-ALONE PROCESS MODEL, IT IS TYPICALLY USED IN SITUATIONS IN WHICH REQUIREMENTS AND/OR THE LOOK-AND-FEEL OF THE USER INTERFACE NEED ADDITIONAL INPUT FROM THE CUSTOMER.
- **BOEHM SPIRAL MODEL** – IS AN ITERATIVE AND EVOLUTIONARY DEVELOPMENT MODEL.
- **UNIFIED PROCESS (UP) MODEL** – IS AN ITERATIVE AND EVOLUTIONARY DEVELOPMENT MODEL.
- **SCRUM AGILE PROCESS MODEL** – IS AN ITERATIVE AND EVOLUTIONARY DEVELOPMENT MODEL.
- **EXTREME PROGRAMMING (XP) AGILE PROCESS MODEL** – IS AN ITERATIVE AND EVOLUTIONARY DEVELOPMENT MODEL.
- **EVOLUTIONARY (EVO) PROJECT MANAGEMENT PROCESS MODEL** – IS AN ITERATIVE AND EVOLUTIONARY DEVELOPMENT MODEL.

COMMONALITY AMONG PROCESS MODELS

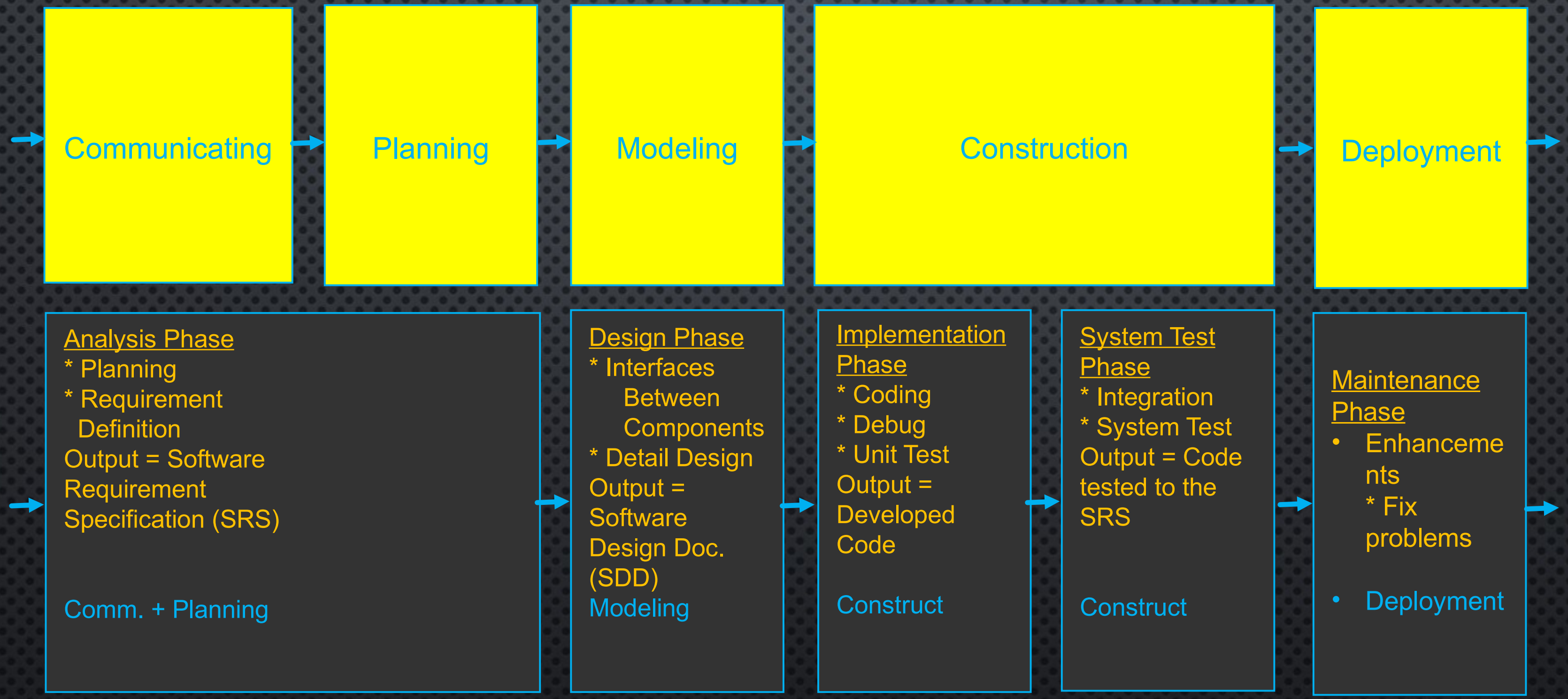
- **COMMUNICATING/PLANNING** – COMMUNICATING/COLLABORATING WITH THE CUSTOMER AND OTHER STAKEHOLDERS, DEFINING THE REQUIREMENTS (THIN SPECIFICATION), AND CREATING A SOFTWARE DEVELOPMENT PLAN DESCRIBING THE WORK, TECHNICAL TASKS, RISKS, RESOURCES, PROCESSES, AND WORK PRODUCTS TO BE PRODUCED WITHIN A SPECIFIC SCHEDULE AND COST ESTIMATE.
- **MODELING** – SELECTING THE BEST DESIGN METHODOLOGY TO PRODUCE A QUALITY WORK PRODUCT ON TIME AND WITHIN BUDGET.
- **CONSTRUCTION** – IMPLEMENTING THE SOLUTION (CODE AND TEST).
- **DEPLOYMENT** – PRODUCT (COMPLETED OR PARTIAL ITERATION) IS DELIVERED TO THE CUSTOMER/STAKEHOLDER WHO EVALUATES THE PRODUCT AND PROVIDES FEEDBACK.

COMMONALITY AMONG PROCESS ACTIVITIES

ALTHOUGH THE TERMINOLOGY BETWEEN PROCESS MODELS IS UNIQUE, THEY HAVE COMPARABLE ACTIVITIES AND PRODUCE SIMILAR WORK PRODUCTS.

12207-2017 ISO/IEC/IEEE Systems and SW Engineering – SW Life-cycle Processes	**Waterfall Life-Cycle Model (Example)**
System Requirements Analysis	Analysis – planning and requirements definition
System Architecture Design	Analysis – planning and requirements definition
Software Requirements Analysis	Analysis – planning and requirements definition
Software Architecture Design	Design – software component interface design
Software Detailed Design	Design – software component internal design
Software Coding and Test	Implementation – software component development and unit test
Software Integration	Software Test – software component integration
Software Qualification Testing	Software Test – software requirements verification
Software Installation	Maintenance – software deployment
Software Acceptance Testing	Maintenance – software system requirements verification and maintenance

PRESCRIPTIVE PROCESS MODEL - WATERFALL



Communicating

Planning

Modeling

Construction

Deployment

Analysis Phase
* Planning
* Requirement Definition
Output = Software Requirement Specification (SRS)

Comm. + Planning

Design Phase
* Interfaces Between Components
* Detail Design
Output = Software Design Doc. (SDD)
Modeling

Implementation Phase
* Coding
* Debug
* Unit Test
Output = Developed Code

Construct

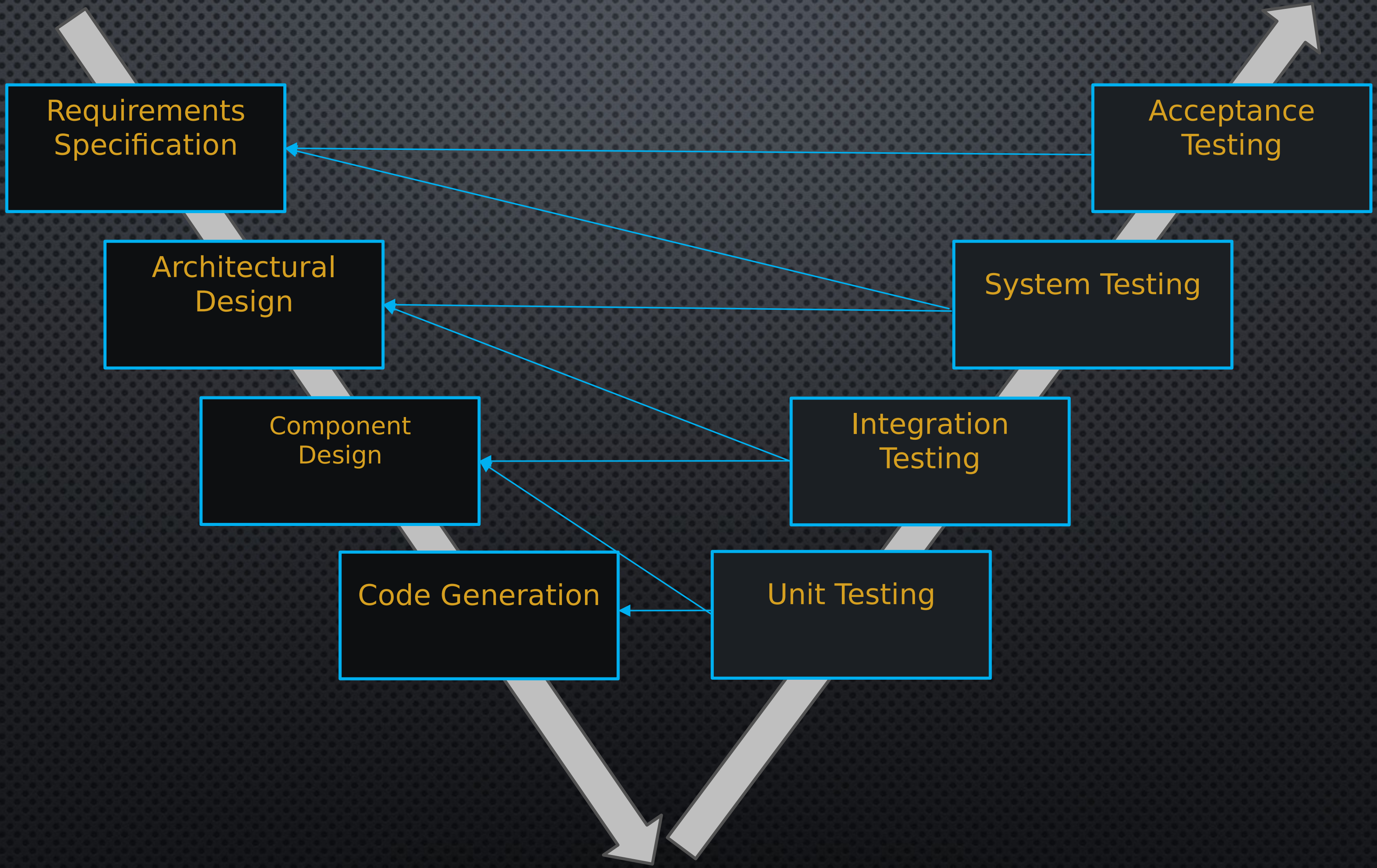
System Test Phase
* Integration
* System Test
Output = Code tested to the SRS

Construct

Maintenance Phase

- Enhanceme nts
  * Fix problems
- Deployment

PRESCRIPTIVE PROCESS MODEL – V-MODEL



Requirements Specification

Architectural Design

Component Design

Code Generation

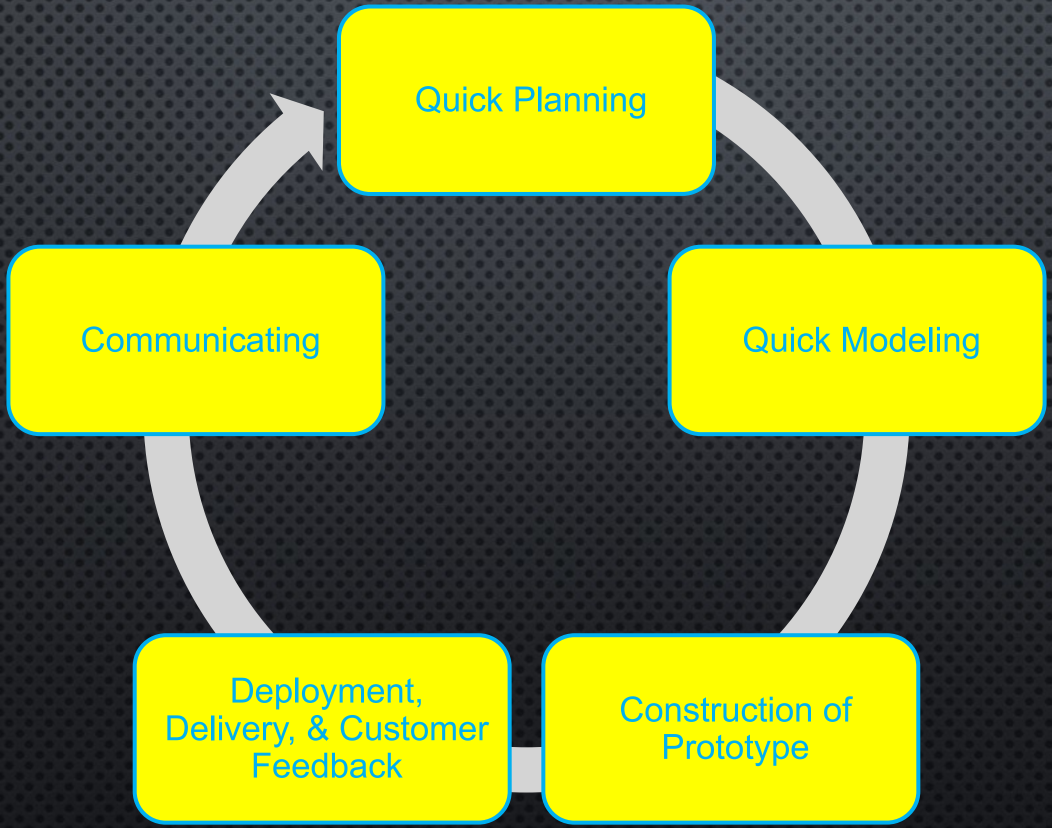
Acceptance Testing

System Testing

Integration Testing

Unit Testing

ITERATIVE AND EVOLUTIONARY PROCESS MODELS



Quick Planning

Quick Modeling

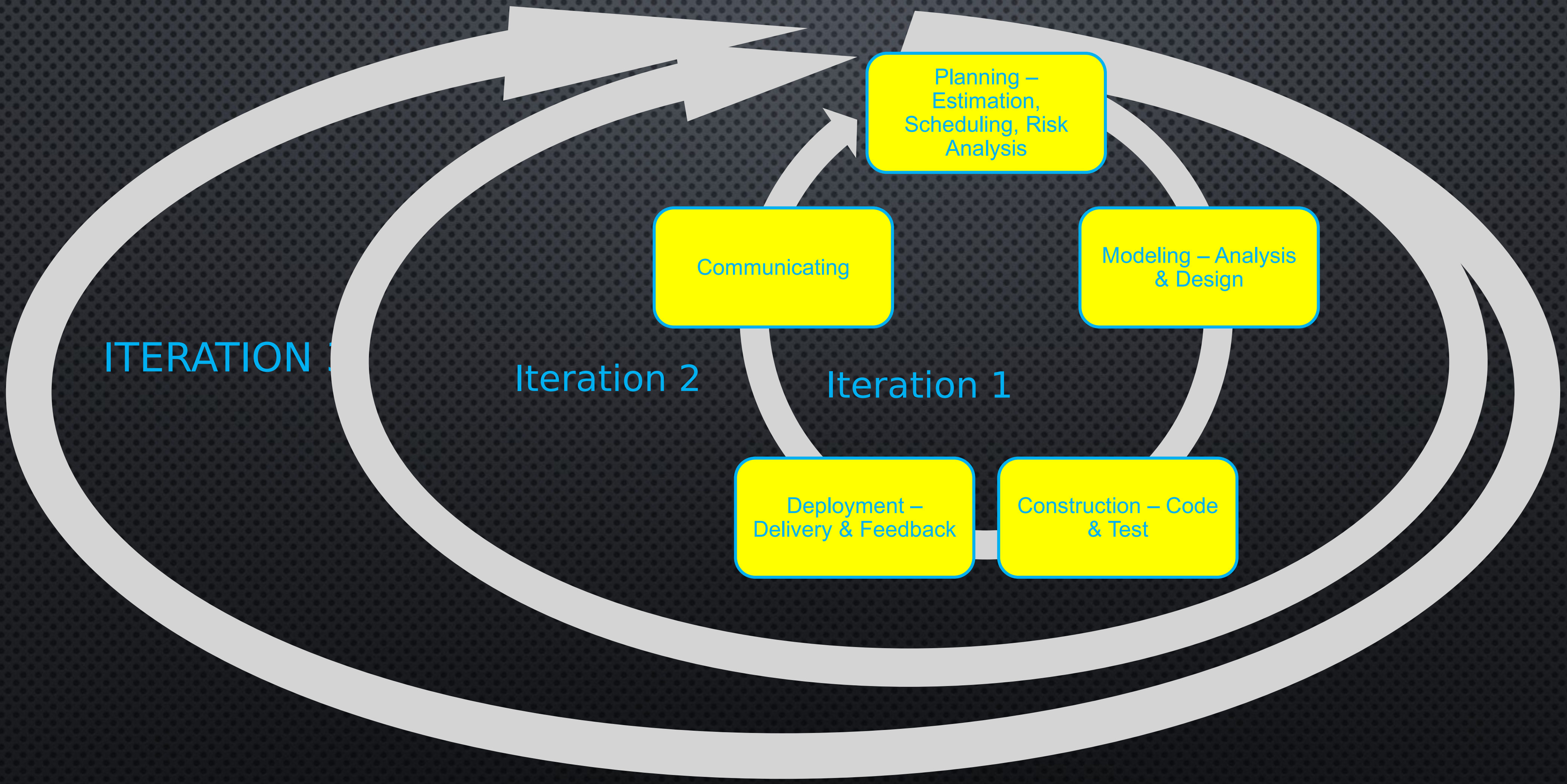
Construction of Prototype

Deployment, Delivery, & Customer Feedback

Communicating

ITERATIVE AND EVOLUTIONARY PROCESS MODELS

BOEHM SPIRAL MODEL


Planning – Estimation, Scheduling, Risk Analysis
Communicating
Modeling – Analysis & Design
Deployment – Delivery & Feedback
Construction – Code & Test
ITERATION 3
Iteration 2
Iteration 1

ITERATIVE AND EVOLUTIONARY PROCESS MODELS

UNIFIED PROCESS (UP) MODEL

SCRUM AGILE PROCESS MODEL

EXTREME PROGRAMMING (XP) AGILE PROCESS MODEL

EVOLUTIONARY (EVO) PROJECT MANAGEMENT PROCESS MODEL

NEW PRODUCT DEVELOPMENT

ACCORDING TO THE AUTHOR, THE "WATERFALL" LIFECYCLE MODEL IS MORE IN LINE WITH "PREDICTABLE MANUFACTURING" IN WHICH PROGRAM PLANNING AND REQUIREMENT SPECIFICATIONS OCCUR UP-FRONT AND IN WHICH ESTIMATES ARE BASED ON KNOWN METHODOLOGIES.

Predictable Manufacturing	New Product Development
Development effort and cost can be determined up front.	Not possible to estimate development effort and cost until empirical data is possible.
Ability to create a detailed schedule containing all the activities that need to be performed.	Ability to create a detailed schedule requires feedback from previous builds.
Adapting to unpredictable change is NOT the NORM. Change rates are relatively low.	Creative adaptation to unpredictable change is the NORM (e.g. Critical Chain, Agile Feedback meeting, etc.) Change rates are high.

NEW PRODUCT DEVELOPMENT

- SOFTWARE TYPICALLY FALLS INTO THE NEW PRODUCT DEVELOPMENT STAGE, ESPECIALLY WHEN NEW TECHNOLOGY IS USED. ITERATIVE AND AGILE METHODS TEND TO BE MORE FLEXIBLE IN MANAGING AND ACHIEVING PROGRAM GOALS.

- HISTORICALLY, THE DEPARTMENT OF DEFENSE (DOD) AND THE U.S. GOVERNMENT HAS BEEN THE LARGEST PURCHASER OF SOFTWARE THROUGH THE 1990’S. MOST GOVERNMENT CONTRACTS FOLLOWED THE WATERFALL LIFECYCLE MODEL OR A DERIVATIVE OF BASED ON THE REQUIREMENTS SPECIFIED IN THE REQUEST FOR PROPOSAL (RFP) THAT A CONTRACTOR RESPONDS TOO. UNFORTUNATELY, MANY OF THE PROGRAMS DURING THE 1970’S AND MID 1980’S OVER RAN DUE TO COST AND SCHEDULE ISSUES AND SOME FAILED TO PRODUCE AN END PRODUCT.

- TODAY THE DOD IS MOVING MORE TOWARDS A DEVOPS ENVIRONMENT IN WHICH THEY MANAGE THE PROJECT INTERNALLY AND HIGHER CONTRACTORS TO PERFORM THE WORK UNDER THEIR DIRECTION WITH ASSISTANCE FROM TECHNICAL EXPORTS FROM THE CUSTOMER. THIS IS ENABLING THE DOD TO MOVE MORE INTO THE AGILE DEVELOPMENT ARENA.

REFERENCES

AGILE & ITERATIVE DEVELOPMENT, A MANAGER’S GUIDE, CRAIG LARMAN, EIGHTH EDITION, ADDISON WESLEY, NEW YORK, NY, COPYRIGHT 2004 BY PEARSON EDUCATION , INC.

CMMI PROCESS MODEL - BY SARAH K. WHITE, SENIOR WRITER, CIO, MAR. 16, 2018

12207-2017 ISO/IEC/IEEE SYSTEMS AND SW ENGINEERING STANDARD

SOFTWARE ENGINEERING, A PRACTITIONER’S APPROACH, EIGHTH EDITION, BY ROGER S. PRESSMAN AND BRUCE R. MAXIM, COPYRIGHT 2015 BY MCGRAW HILL, NEW YORK, NY

PROCESS

AGILE & ITERATIVE DEVELOPMENT (CHAPTER 1)

A MANAGER’S GUIDE BY: CRAIG LARMAN

WEEK 2: INTRODUCTION – THE KEY TO SUCCESSFUL PROGRAM MANAGEMENT

BY: JOSEPH MARTINAZZI

LEAD WITH HONESTY AND INTEGRITY

PROGRAM MANAGEMENT IS RESPONSIBLE FOR DEFINING CLEAR OBJECTIVES FOR THE PROGRAM TO ENSURE THE END-PRODUCT WILL MEET OR EXCEED THE CUSTOMER'S EXPECTATION (RELIABILITY, SECURITY, AND SAFETY).

YOUR JOB AS A SOFTWARE MANAGER IS TO NOT ONLY UNDERSTAND PROGRAM OBJECTIVES, BUT TO ALWAYS BE LOOKING AHEAD IN YOUR PLANNING SO THAT YOUR TEAM CAN BE ADAPTIVE AND NOT REACTIVE WHEN ISSUES ARISE.

TO BE A SUCCESSFUL LEADER YOU NEED TO:

- EFFECTIVELY COMMUNICATE TO BOTH MANAGEMENT AND YOUR TEAM,
- DON'T CREATE A PLAN THAT REQUIRES YOUR TEAM TO ALWAYS WORK AT 110% (*SAVE IT FOR WHEN MURPHY'S LAW HITS*),
- UTILIZE YOUR STAFF TO THE BEST OF THEIR ABILITIES *(EVERYONE CAN BE A VALUABLE CONTRIBUTOR),*
- CREATE A TEAM ENVIRONMENT *(YOU SUCCEED OR FAIL AS A TEAM),*
- LEAD BY EXAMPLE (*DON'T ASK YOUR TEAM TO DO SOMETHING YOU WOULDN'T DO YOURSELF*),
- BECOME CAPTAIN AMERICA - SHIELD YOUR TEAM FROM PROGRAM POLITICS
- HAVE AN OUTWARDLY POSITIVE ATTITUDE!

EVERYTHING IS AWESOME, EVERYTHING IF COOL WHEN YOUR PART OF JOE'S TEAM. EVERYTHING IS AWESOME WHEN YOUR LIVING THE PROGRAM X-Y-Z DREAM !

WHAT DO WE HAVE TO DO - KEEP MOVING FORWARD!

INITIAL PROGRAM PLANNING

INTEGRATED PROGRAM MANAGEMENT

PROGRAM ORGANIZATION

KEY PROGRAM PROCESSES:
1. PROGRAM REQUIREMENTS DOCUMENT,
2. PROGRAM CHANGE MANAGEMENT PLAN,
3. PROGRAM VERIFICATION & VALIDATION PLAN,

4. PROGRAM SCHEDULE AND MILESTONES (IMP AND IMS),
5. PROGRAM WORK BREAKDOWN STRUCTURE/COST COLLECTION METHOD,

6. PROGRAM STAKE HOLDER INVOLVEMENT PLAN,
7. PROGRAM QUALITY ASSURANCE PLAN, AND
8. PROGRAM RISK MANAGEMENT PLAN.

As part of the initial program planning effort, key processes must be established that take the following into account:

- The contractual requirements the system under development must meet.
- A program "Change Management Plan" that describes how program artifacts will be controlled and how change will be managed during the development of the product (in-house) as well as for the delivered product (customer facility) if necessary.
- The overall program testing strategy based on the contract and how the end-products will be sold off to the customer.
- The time period (schedule) in which the system must be completed including any interim milestones that must be met.
- The budget allocated to the effort you will be responsible for managing. *This includes task your team is directly responsible for as well as support tasks for other organizations.*
- The facilities (lab environment) you will need to support the development of the end-product.
- A staffing plan to efficiently staff the program to meet cost and schedule requirements.
- Risks associated with achieving program goals for the tasks your team has been allocated.

INTEGRATED MASTER PLAN & MASTER SCHEDULE

A PROGRAM SCHEDULE IS TYPICALLY REFERRED TO AS AN INTEGRATED MASTER SCHEDULE (IMS) AND IS BASED OFF A PROGRAM INTEGRATED MASTER PLAN (IMP) THAT WAS CREATED AT THE TIME OF THE PROPOSAL.

ACCORDING TO THE “MITRE SYSTEMS ENGINEERING GUIDE”, A DEFINITION FOR AN IMP AND AN IMS ARE AS FOLLOWS:

- *“THE IMP COMPRISES A HIERARCHY OF PROGRAM EVENTS, IN WHICH EACH EVENT IS SUPPORTED BY SPECIFIC ACCOMPLISHMENTS, AND EACH ACCOMPLISHMENT IS BASED ON SATISFYING SPECIFIC CRITERIA TO BE CONSIDERED COMPLETE. THE IMP IS AN EVENT-DRIVEN PLAN IN WHICH THE EVENTS ARE NOT TIED TO CALENDAR DATES; THEY ARE TIED TO THE ACCOMPLISHMENT OF A TASK OR WORK PACKAGE AS EVIDENCED BY THE SATISFACTION OF THE SPECIFIED CRITERIA FOR THAT ACCOMPLISHMENT.”*
- *“THE IMS IS AN INTEGRATED, NETWORKED SCHEDULE OF ALL THE DETAILED, DISCRETE WORK PACKAGES AND PLANNING PACKAGES (OR LOWER LEVEL TASKS OF ACTIVITIES) NECESSARY TO SUPPORT THE IMP'S EVENTS, ACCOMPLISHMENTS, AND CRITERIA. THE IMS IS DEVELOPED FROM THE IMP, MAJOR CONTRACTOR EVENTS, ACCOMPLISHMENTS, ENTRANCE CRITERIA, EXIT CRITERIA, AND THE WBS, WHICH DEFINES THE PROGRAM WORK STRUCTURE AND WORK PACKAGES. THE IMS IS TIME DRIVEN, TIED TO CALENDAR DATES, AND SHOULD BE DEFINED TO THE LEVEL OF DETAIL NECESSARY FOR PROGRAM EXECUTION.”*

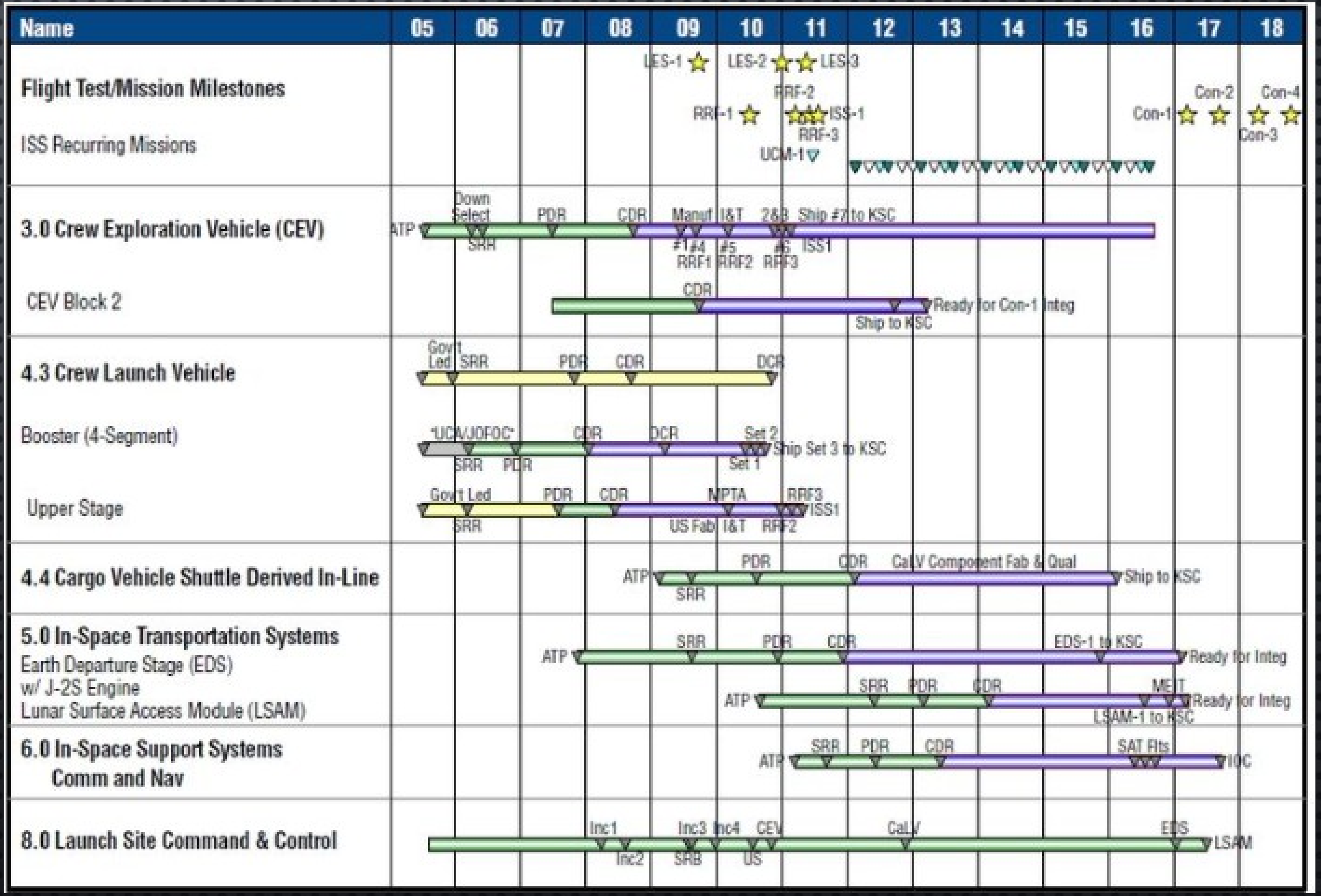
REFERENCES AND RESOURCES
DAU, DEFENSE ACQUISITION GUIDEBOOK, CHAPTER 11.3.1.4.2, INTEGRATED MASTER SCHEDULE (IMS), ACCESSED JUNE 7, 2016.
DEPARTMENT OF DEFENSE, OCTOBER 21, 2005, INTEGRATED MASTER PLAN AND INTEGRATED MASTER SCHEDULE: PREPARATION AND USE GUIDE, VER. 0.9, ACCESSED SEPTEMBER 14, 2017.
INCOSE, 2015, SYSTEMS ENGINEERING HANDBOOK, A GUIDE FOR SYSTEM LIFE-CYCLE PROCESSES AND ACTIVITIES, FOURTH ED., INCOSE-TP-2003-002-04.
PROJECT MANAGEMENT INSTITUTE (PMI), 2013, STANDARD FOR PROGRAM MANAGEMENT, THIRD ED

HTTPS://WWW.MITRE.ORG/PUBLICATIONS/SYSTEMS-ENGINEERING-GUIDE/ACQUISITION-SYSTEMS-ENGINEERING/ACQUISITION-PROGRAM-PLANNING/INTEGRATED-MASTER-SCHEDULE-IMSINTEGRATED-MASTER-PLAN-IMP-APPLICATION

INTEGRATED MASTER PLAN & MASTER SCHEDULE

EXAMPLE IMP AND IMS

The high-level program schedule typically comes directly from the contract and is based on the proposed schedule in the Request for Proposal (RFP) that was refined in your company’s proposal.



Typepad – Creative Commons

INTEGRATED MASTER PLAN & MASTER SCHEDULE

THE IMS IS ONE OF THE MOST IMPORTANT PROGRAM MANAGEMENT TOOLS TO EFFECTIVELY MANAGING THE PROGRAM.

- IT IS USED TO REPORT PROGRESS TO THE CUSTOMER, AND
- IT IS USED BY PROGRAM MANAGEMENT TO DETERMINE IF THE PROGRAM IS ON SCHEDULE.

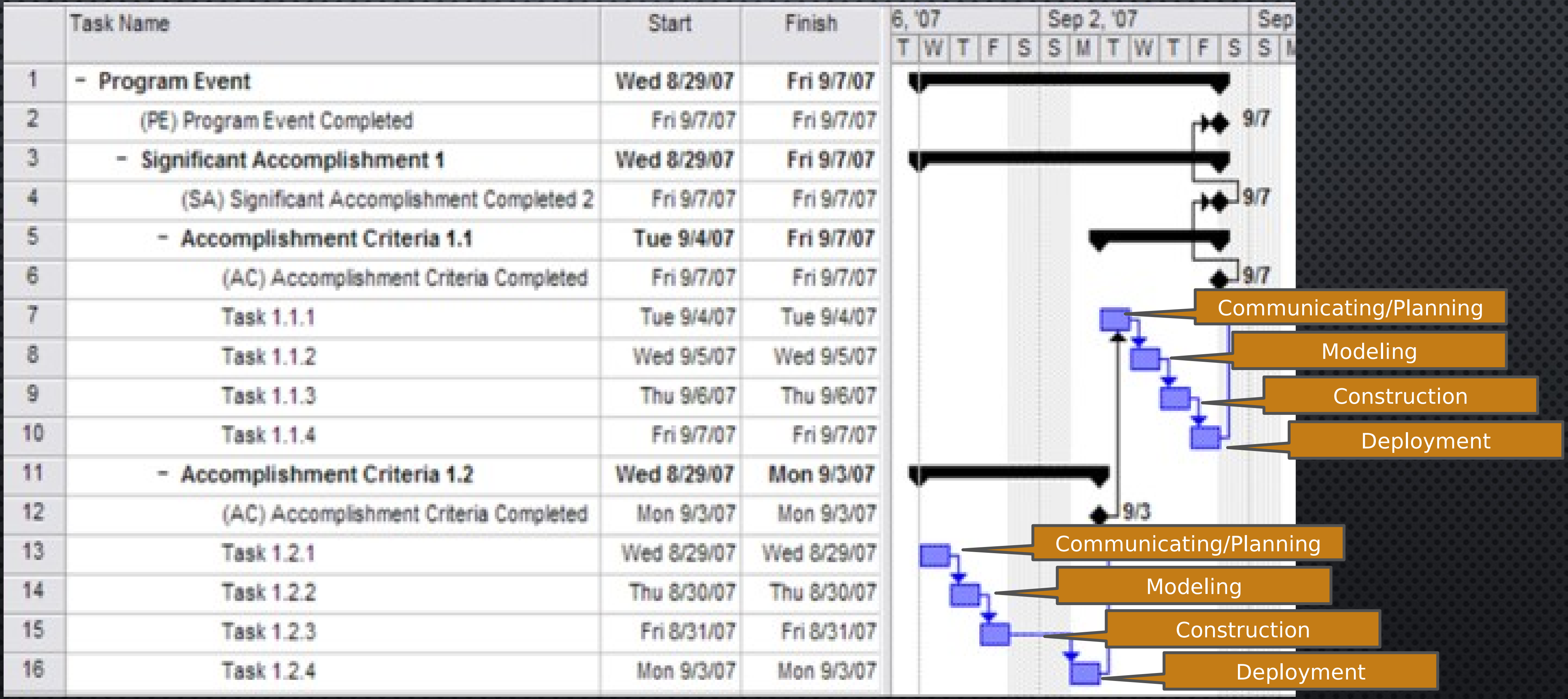
AT THE END OF STARTUP PLANNING THE IMS WILL BE BASELINED WITH ALL PROGRAM TASKS IDENTIFIED IN EITHER A

- WORK PACKAGE (IMPLYING THE TASK CAN BE WORKED WITHIN THE NEXT SEVERAL MONTHS) OR IN A
- PLANNING PACKAGE (TASKS TO BE WORKED AT SOME FUTURE DATE ON THE PROGRAM).
- SOME WORK MAY ALSO BE PLANNED AS A LEVEL OF EFFORT (LOE).
  MY RECOMMENDATION IS TO AVOID USING LOE.

EACH TASK WILL ALSO HAVE

- AN ASSOCIATED DURATION (HOURS),
- AN ASSOCIATED BUDGET (DOLLARS),
- AND A MEASURE OF HOW STATUS WILL BE REPORTED (E.G. 50-50, PERCENT COMPLETE, ETC.).
  MY RECOMMENDATION IS TO ALWAYS DEFAULT TO USING THE PERCENT COMPLETE METHOD AS THE PREFERRED METHOD TO MEASURE PROGRESS ON A TASK.

INTEGRATED MASTER PLAN & MASTER SCHEDULE



	Task Name	Start	Finish
1	- Program Event	Wed 8/29/07	Fri 9/7/07
2	(PE) Program Event Completed	Fri 9/7/07	Fri 9/7/07
3	- Significant Accomplishment 1	Wed 8/29/07	Fri 9/7/07
4	(SA) Significant Accomplishment Completed 2	Fri 9/7/07	Fri 9/7/07
5	- Accomplishment Criteria 1.1	Tue 9/4/07	Fri 9/7/07
6	(AC) Accomplishment Criteria Completed	Fri 9/7/07	Fri 9/7/07
7	Task 1.1.1	Tue 9/4/07	Tue 9/4/07
8	Task 1.1.2	Wed 9/5/07	Wed 9/5/07
9	Task 1.1.3	Thu 9/6/07	Thu 9/6/07
10	Task 1.1.4	Fri 9/7/07	Fri 9/7/07
11	- Accomplishment Criteria 1.2	Wed 8/29/07	Mon 9/3/07
12	(AC) Accomplishment Criteria Completed	Mon 9/3/07	Mon 9/3/07
13	Task 1.2.1	Wed 8/29/07	Wed 8/29/07
14	Task 1.2.2	Thu 8/30/07	Thu 8/30/07
15	Task 1.2.3	Fri 8/31/07	Fri 8/31/07
16	Task 1.2.4	Mon 9/3/07	Mon 9/3/07

Typepad – Creative Commons

INTEGRATED MASTER PLAN & MASTER SCHEDULE

PROGRAM STATUS CAN BE COLLECTED FROM AN EARNED VALUE MANAGEMENT SYSTEM AND PULLED INTO THE IMS ON A WEEKLY BASIS TO DETERMINE THE PROGRESS OF ALL TASKS CURRENTLY IN WORK.

- TASKS WILL EITHER SHOW THEY ARE AHEAD OF SCHEDULE, ON SCHEDULE, OR BEHIND SCHEDULE.
- ON A ROLL-UP LEVEL, THIS WILL SHOW IF THE OVERALL PROGRAM IS AHEAD, ON, OR BEHIND SCHEDULE. ***IF THE PROGRAM SCHEDULE IS DELAYED; COSTS BECOME UNCONTROLLABLE.***

REMEMBER PROGRAM BUDGETS ARE CALENDARIZED BASED ON THE IMS. AS A RESULT, IF SEVERAL OF THE TASKS YOU ARE RESPONSIBLE FOR SLIP INTO A DIFFERENT CALENDAR YEAR, THEY MAY OVER-RUN BASED ON DOLLARS EVEN IF THEY COMPLETE WITHIN THE HOURS ALLOCATED TO COMPLETE THE WORK ASSOCIATED WITH THE TASK.

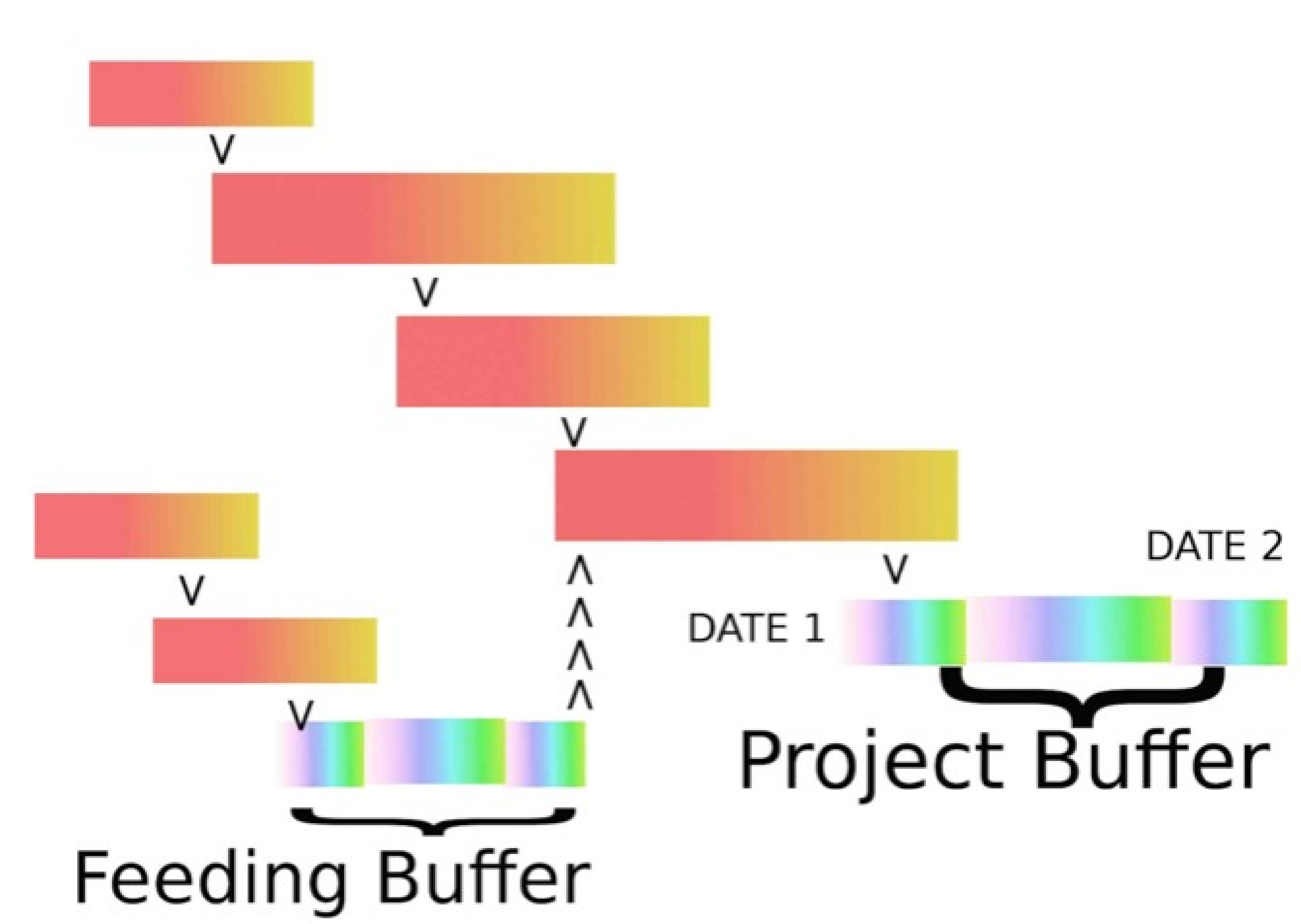
THE ABILITY TO MONITOR ALL TASKS WITHIN A SCHEDULE GIVES PROGRAM MANAGEMENT THE ABILITY TO DETERMINE CRITICAL PATHS THAT DRIVE THE PROGRAM. TYPICALLY, PROGRAMS WILL MONITOR THE TOP THREE CRITICAL PATHS THROUGH THE PROGRAM. TYPICALLY SOFTWARE DEVELOPMENT, INTRA-SITE COMMUNICATION DEVELOPMENT, OR HARDWARE PROCUREMENT WILL DRIVE THE PROGRAM CRITICAL PATHS.

IF A TASK YOU ARE RESPONSIBLE FOR IS ON ONE OF THE PROGRAM CRITICAL PATHS YOU WILL NEED TO BE PREPARED TO ALWAYS HAVE A SOLUTION AS TO HOW TO MINIMIZE THE SCHEDULE TO COMPLETE THAT TASK.

MANAGING PROGRAM SCHEDULE BUFFERS

AN EFFECTIVE WAY TO MINIMIZE PROGRAM SCHEDULE IMPACT AND KEEP INSIGHT INTO TASK IMPLEMENTATION IS BY UTILIZING A PROCESS KNOWN AS “CRITICAL CHAIN PROJECT MANAGEMENT” WHICH FOCUSES ON MANAGING PROGRAM BUFFERS AND RESOURCES (STAFFING AND LAB EQUIPMENT).

PROGRAMS THAT SUCCESSFULLY MANAGE THEIR BUFFERS WILL END UP BEING SUCCESSFUL.



WordPress.com – Creative Commons

MANAGING PROGRAM SCHEDULE BUFFERS

SW DEVELOPERS ARE ALWAYS OPTIMISTIC WHEN ASKED HOW LONG IT WILL TAKE TO COMPLETE A TASK. THEIR ANSWER NEVER ACCOUNTS FOR MURPHY’S LAW (IF SOMETHING CAN GO WRONG IT WILL GO WRONG).

AS A MANAGER YOU NEED TO ACCOUNT FOR MURPHY’S LAW BY MANAGING THE DEVELOPMENT SCHEDULE. ONE OF THE MOST EFFECTIVE WAYS TO DO THIS IS TO CREATE A BUILD PLAN WITH A SCHEDULE BUFFER BUILT IN.

FOR EXAMPLE, IF YOU SAY BUILD 1 WILL TAKE 3 MONTHS TO COMPLETE IN THE BUILD PLAN.

- WHEN YOU ACTUALLY PLAN THE ACTIVITY IN THE IMS YOU WILL NEED TO BUILD IN A 3 WEEK BUFFER BY ACTUALLY PLANNING ALL OF THE TASKS TO COMPLETE 3 WEEKS AHEAD OF THE ADVERTISED FINISH OF THE BUILD BY USING SCHEDULE DEPENDENCIES (BASED ON RESOURCE AVAILABILITY) VS. TECHNICAL DEPENDENCIES (BASE ON COMPONENT DESIGN DEPENDENCIES).
- ANOTHER WAY TO ACHIEVE THIS IS TO HAVE TASKS TO FIX ISSUES ASSOCIATED WITH EACH BUILD THAT THE DEVELOPMENT WORK FEEDS INTO.

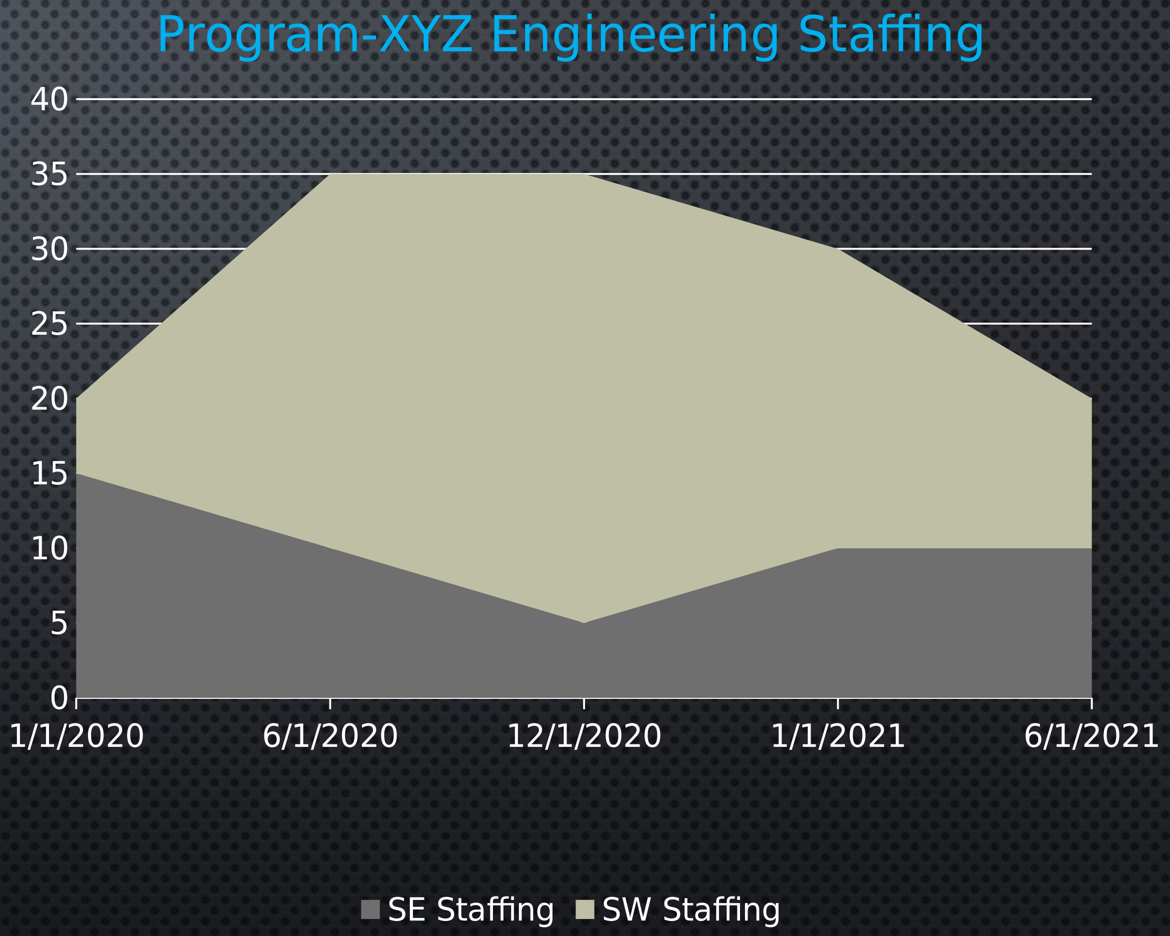
PROGRAM STAFFING NEEDS

THE IMS IS ALSO A KEY TOOL TO IDENTIFY IF PROGRAM STAFFING WILL IMPACT PRODUCT DEVELOPMENT FROM BOTH A SCHEDULE AND COST PERSPECTIVE.

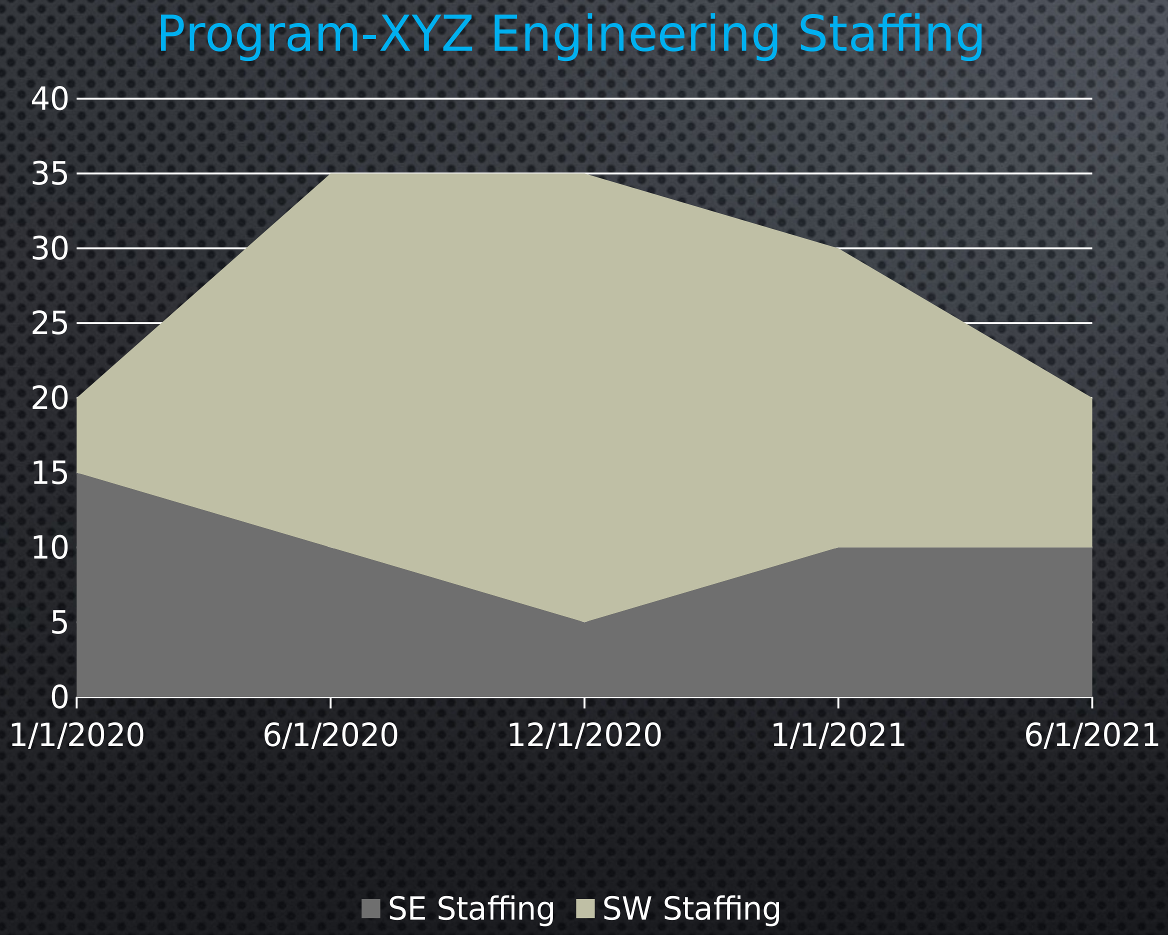
SINCE THE IMS PROVIDES A CALENDARIZED ROLE UP AS TO WHEN TASKS NEED TO BE COMPLETED (HOURS) A STAFFING PROFILE CAN BE EXTRACTED FROM THE IMS WHICH INDICATES HOW MANY PEOPLE WILL BE NEEDED TO COMPLETE THE WORK AND THE DURATION THOSE INDIVIDUALS WILL BE NEEDED FOR.

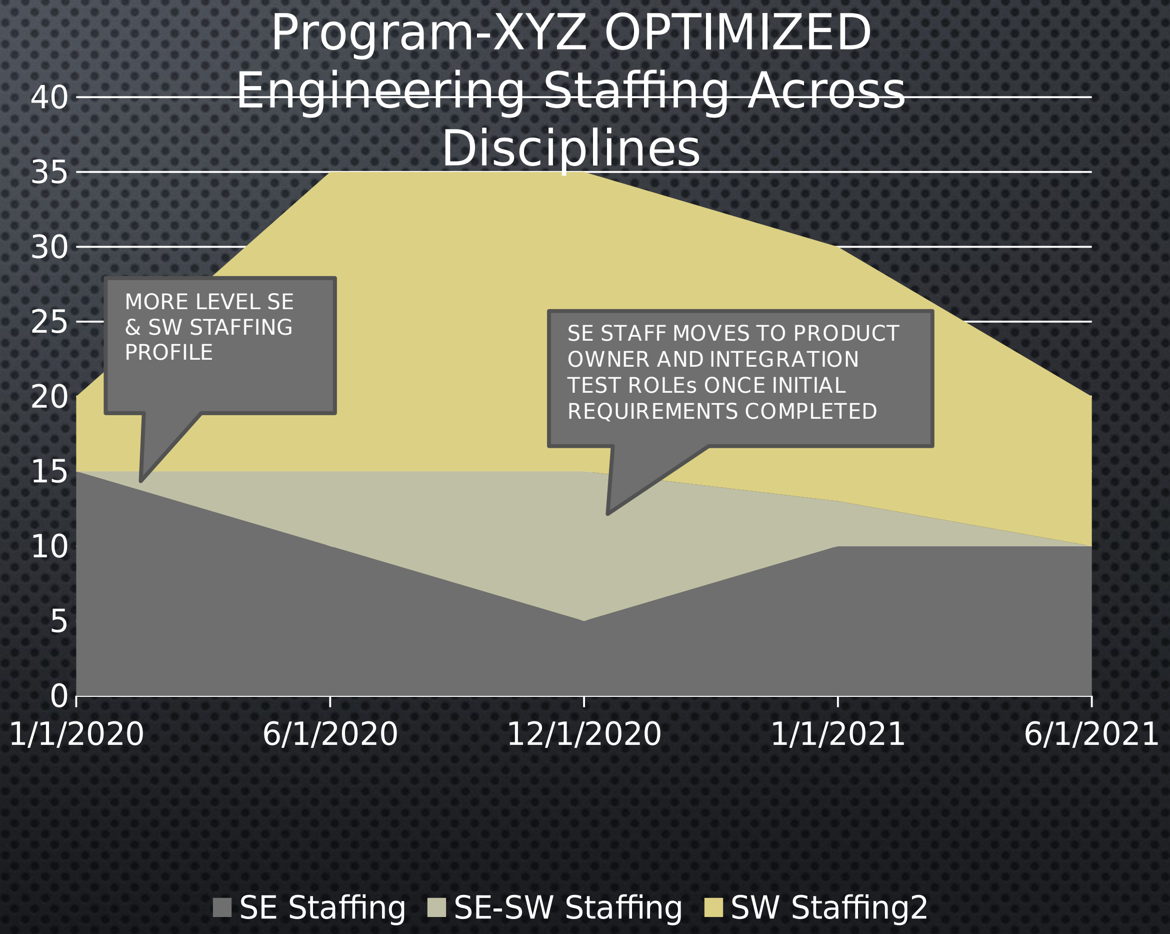
THIS STAFFING PROFILE WILL INDICATE HOW MANY PEOPLE YOU NEED TO COMPLETE THE TASKS WITHIN YOUR SCHEDULE.

- IF YOUR ORGANIZATION IS OVER STAFFED, YOU SHOULD BE COMPLETING TASKS AHEAD OF SCHEDULE TO STAY ON BUDGET.
- IF YOUR ORGANIZATION IS CORRECTLY STAFFED, YOU SHOULD BE COMPLETING TASK ON OR AHEAD OF SCHEDULE (ASSUMING THE TASKS ARE CORRECTLY SCOPED),
- IF YOUR ORGANIZATION IS UNDERSTAFFED YOU COULD END UP IN A SITUATION IN WHICH THE ENTIRE PROGRAM SCHEDULE FALLS BEHIND IF YOU CAN NOT COMPLETE THE WORK AS PLANNED.



PROGRAM STAFFING NEEDS

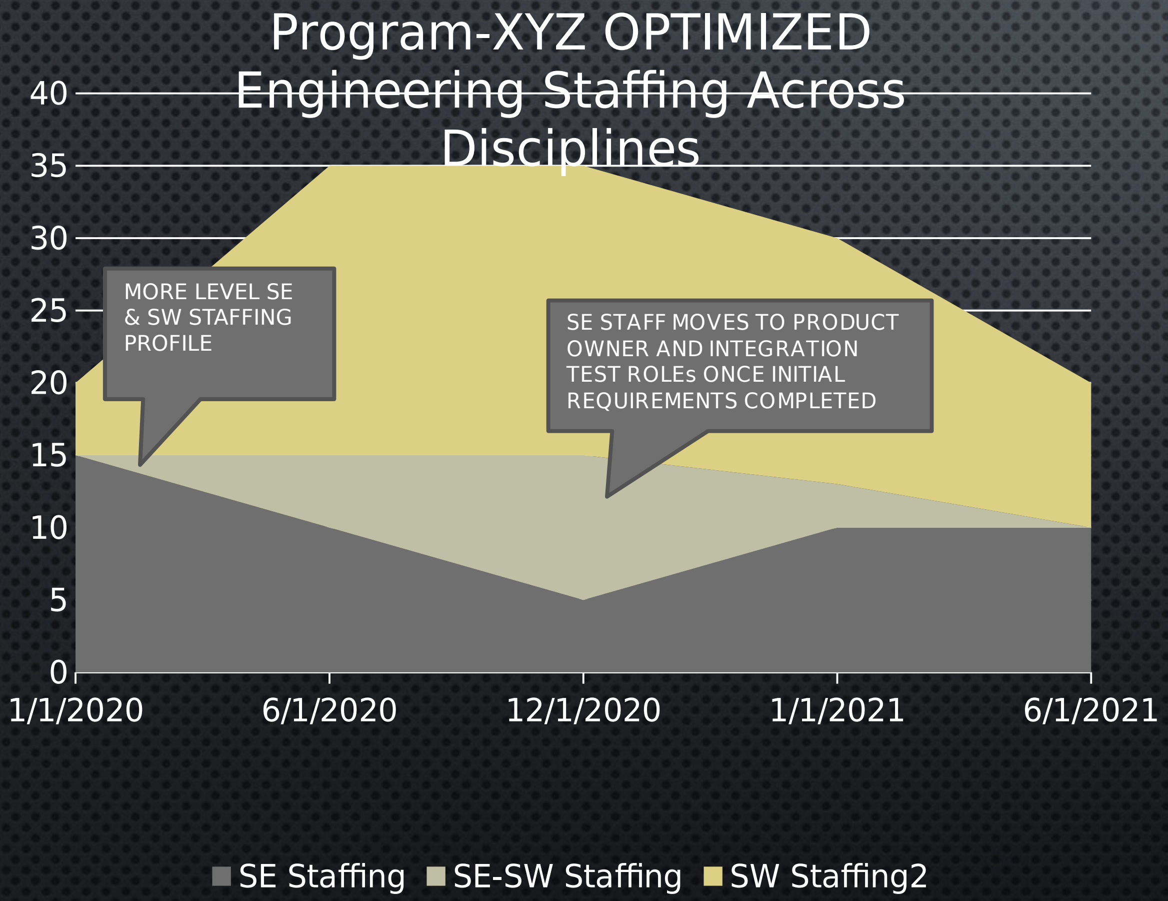


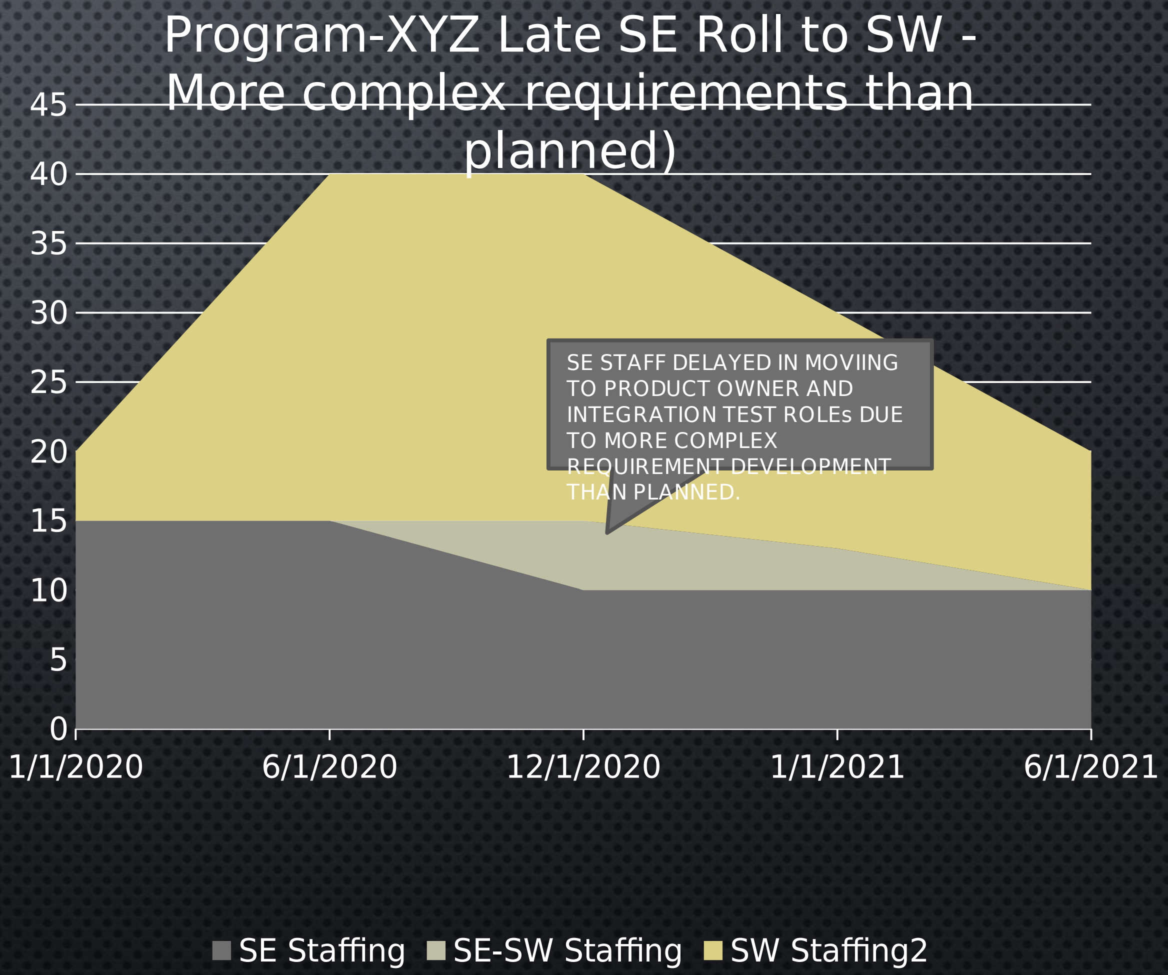


MORE LEVEL SE & SW STAFFING PROFILE

SE STAFF MOVES TO PRODUCT OWNER AND INTEGRATION TEST ROLEs ONCE INITIAL REQUIREMENTS COMPLETED

PROGRAM STAFFING NEEDS





SW PROJECT PLANNING - OVERVIEW

THE INITIAL PLANNING STAGE OF A PROGRAM HAPPENS WITHIN A VERY SHORT PERIOD OF TIME.

THE KEY OBJECTIVES FOR THE SOFTWARE PROGRAM MANAGER (SPM) DURING THIS PHASE OF THE PROGRAM INCLUDES:

- DETAIL PLANNING TASKS THAT NEED TO BE COMPLETED WITHIN THE PROGRAM SCHEDULE.
- ACCOUNTING FOR THE ASSOCIATED RISKS IN THE DEVELOPMENT OF THOSE TASKS,
- DETERMINING IF THOSE RISKS NEED TO BE REALIZED OR MITIGATED AS PART OF THE INITIAL PLAN,
- SECURING THE NECESSARY BUDGET TO COMPLETE ALL TASKS ASSIGNED TO THE ORGANIZATION,
- SECURING THE SCHEDULE NEEDED TO DEVELOP THE PRODUCT, AND
- CREATING ALL STARTUP PROCESSES NEEDED TO EXECUTE THE SOFTWARE DEVELOPMENT EFFORT
- SPENDING AS LITTLE MONEY AS POSSIBLE DURING THIS PHASE OF THE PROGRAM.

SW PROJECT PLANNING - ASSUMPTIONS

FOR THE PURPOSE OF THIS DISCUSSION, WE ARE GOING TO ASSUME

- THE COMPANY YOU WORK FOR HAS AN EXISTING PRODUCT THEY PLAN ON LEVERAGING TO MEET MOST OF THE REQUIREMENTS SPECIFIED IN THE CUSTOMER’S RFP.
- THIS PRODUCT HAS AN ASSOCIATED SOFTWARE BASELINE THAT CAN BE LEVERAGED AS A STARTING POINT FOR THE PROGRAM.
- SINCE YOUR COMPANY HAS A CMMI LEVEL 5 RATING, WE KNOW THAT THE COMPANY HAS ESTABLISHED METRICS ON BOTH SOFTWARE SIZE AND SOFTWARE PRODUCTIVITY THAT WENT INTO THE PROPOSAL ESTIMATE (MEANING THE BUDGET YOUR ORGANIZATION BID SHOULD BE SUFFICIENT TO COVER THE COST OF IMPLEMENTING THE SYSTEM).

SW PROJECT PLANNING – INITIAL STAFFING

THE SOFTWARE STAFF DURING THIS STAGE OF THE PROGRAM NEEDS TO BE MINIMALIZED TO PREVENT EROSION OF BUDGET DURING THE PLANNING STAGE.

> *MY RECOMMENDATION IS THAT IT INCLUDES THE SPM, A SOFTWARE TECHNICAL LEAD (STD) AND ADDITIONAL STAFF ONLY AS NEEDED TO SUPPORT THE PROGRAM STARTUP ACTIVITIES.*

REMEMBER THE JOB OF THE SPM IS TO ALWAYS LOOK AHEAD AND MITIGATE PROGRAM RISK.

- YOUR SUCCESS WILL ULTIMATELY BE DETERMINED BASED ON YOUR ABILITY TO COMPLETE THE PROGRAM WITHIN COST AND SCHEDULE.
- THE ONLY ONE THAT WILL REMEMBER CHALLENGES YOU HAVE TO OVERCOME IS YOURSELF!
- CHALLENGES MAY COME FROM THE PROGRAM OFFICE (PMO) OR FROM YOU OWN ORGANIZATION WITHIN THE COMPANY. INITIAL PROGRAM STAFFING MAY BE THE FIRST CHALLENGE YOU HAVE TO OVERCOME.
  - IF YOUR ORGANIZATION FORCES YOU TO STAFF UP EARLIER THAN YOU WANT, MAKE SURE YOU HAVE A WAY TO ACCURATELY COLLECT COSTS, HAVE TASKS INDIVIDUALS CAN WORK TO ACHIEVE PROGRAM GOALS, AND HAS MANAGEMENT OVERSITE.
  - IF YOUR ORGANIZATION DOESN’T HAVE THE STAFF TO SUPPORT YOUR PROGRAM, BRING SOMEONE ON BOARD WHO IS CAPABLE OF OPENING JOB REQUISITIONS INTERVIEWING AND GETTING THE STAFFING TALENT YOUR PROGRAM WILL NEED.

SW PROJECT PLANNING – ESTABLISHING A BASELINE SCHEDULE AND BUDGET

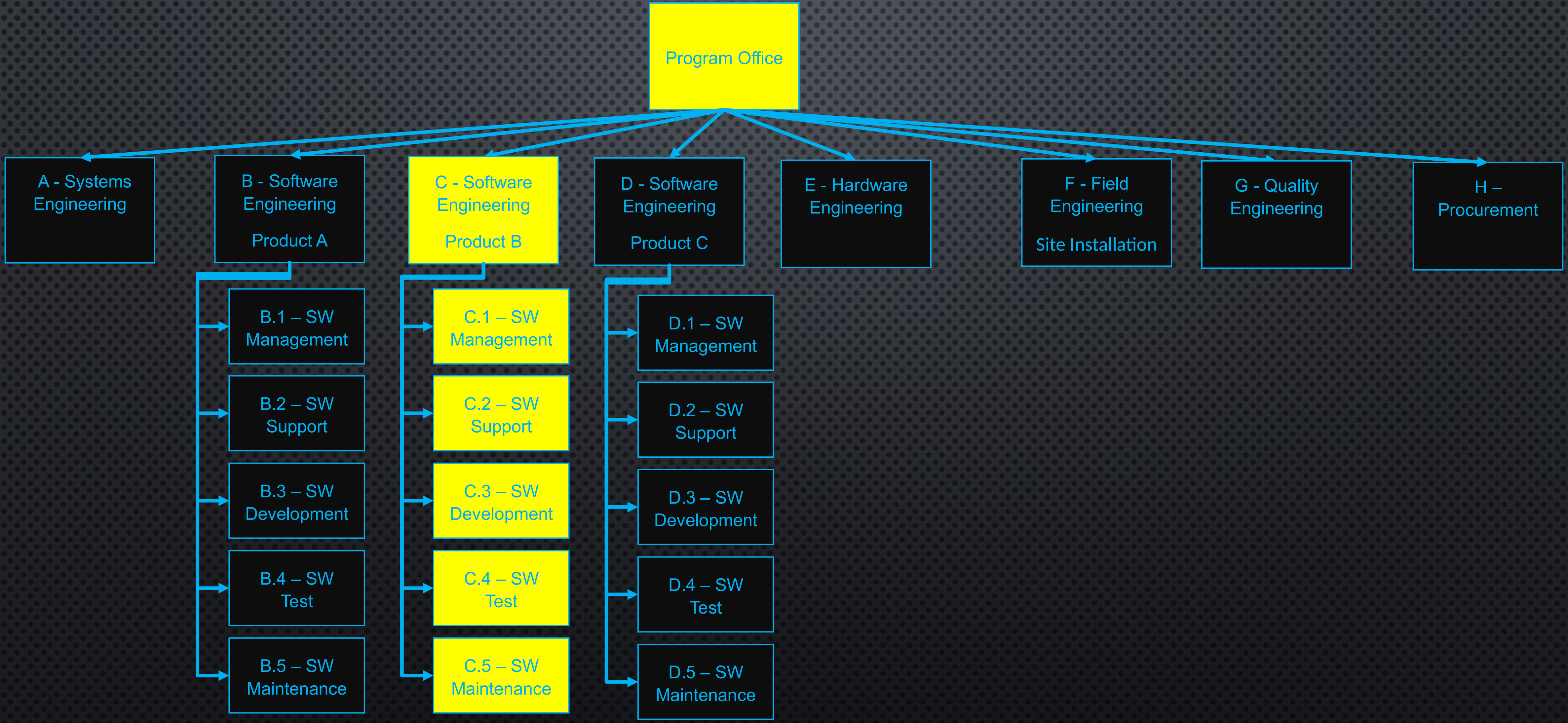
YOUR GOAL AS A SOFTWARE MANAGER WILL BE TO UNDERSTAND THE IMP AND IMS AND CREATE A SOFTWARE BUILD PLAN (SBP) THAT CAN FIT INTO THE PROGRAM SCHEDULE THAT ACCOUNTS FOR ALL THE TASKS YOUR TEAM WILL BE RESPONSIBLE TO COMPLETE OR SUPPORT.

IN ADDITION, YOU WILL BE RESPONSIBLE FOR UNDERSTANDING THE SOFTWARE BID YOUR ORGANIZATION PUT FORTH IN RESPONSE TO THE CUSTOMER'S RFP AND JUSTIFY THE COST TO COMPLETE EACH ACTIVITY TO THE PROGRAM OFFICE PRIOR TO THEM ALLOCATING YOUR BUDGET.

ALL ACTIVITIES WITHIN THE PROPOSAL ARE ORGANIZED IN A WORK BREAKDOWN STRUCTURE (WBS) THAT MAPS HOURS AND BUDGETS TO ACTIVITIES THAT NEED TO BE PERFORMED AS PART OF PROGRAM EXECUTION. THESE ACTIVITIES INCLUDE:

- SOFTWARE MANAGEMENT ACTIVITIES,
- SOFTWARE SUPPORT ACTIVITIES,
- SOFTWARE DEVELOPMENT ACTIVITIES,
- SOFTWARE TEST ACTIVITIES, AND
- SOFTWARE MAINTENANCE ACTIVITIES

SW PROJECT PLANNING – EXAMPLE PROGRAM WBS



- Program Office
  - A - Systems Engineering
  - B - Software Engineering Product A
    - B.1 – SW Management
    - B.2 – SW Support
    - B.3 – SW Development
    - B.4 – SW Test
    - B.5 – SW Maintenance
  - C - Software Engineering Product B
    - C.1 – SW Management
    - C.2 – SW Support
    - C.3 – SW Development
    - C.4 – SW Test
    - C.5 – SW Maintenance
  - D - Software Engineering Product C
    - D.1 – SW Management
    - D.2 – SW Support
    - D.3 – SW Development
    - D.4 – SW Test
    - D.5 – SW Maintenance
  - E - Hardware Engineering
  - F - Field Engineering Site Installation
  - G - Quality Engineering
  - H – Procurement

SW PROJECT PLANNING – ESTABLISHING A BASELINE SCHEDULE AND BUDGET

IF YOU’RE LUCKY, AN ARTIFACT THAT MAY HAVE BEEN CREATED DURING THE PROPOSAL PHASE IS A THIN SPECIFICATION (SPEC.).

- THIS DOCUMENT TYPICALLY GOES ALONG WITH A BID AN ORGANIZATION PUT FORTH THAT ESTIMATED HOW MUCH EFFORT IT WOULD TAKE TO IMPLEMENT THE CAPABILITIES/FEATURES OF THE END-PRODUCT SPECIFIED IN THE RFP.

DURING STARTUP, THE PROPOSAL INFORMATION IS USED TO:

1. CREATE A SOFTWARE BUILD PLAN (SBP),
2. SUPPORT DETAILED IMS PLANNING,
3. JUSTIFY YOUR ORGANIZATIONS BUDGET TO PROGRAM MANAGEMENT,
4. SOLIDIFY YOUR THE WBS STRUCTURE YOU WANT TO USE TO MANAGE THE PROGRAM, AND
5. CREATE A SOFTWARE DEVELOPMENT PLAN (SDP).

SW PROJECT PLANNING – CREATING THE SBP

1. CREATE A SOFTWARE BUILD PLAN (SBP)

   THE SPM NEEDS TO WORK WITH THE STD IN LAYING OUT THE PRELIMINARY SBP WHICH SIMPLY BREAKS THE SOFTWARE INTO MAJOR CAPABILITIES OR FEATURES AND IDENTIFIES THE SEQUENCE IN WHICH THOSE CAPABILITIES/FEATURES WILL BE DEVELOPED (FROM AN AGILE PERSPECTIVE YOU CAN THINK OF THIS EXERCISE AS CREATING A PRODUCT BACKLOG AND ESTIMATING THE EFFORT ASSOCIATED WITH DEVELOPING THESE HIGH-LEVEL FEATURES)

   1) THE FIRST STEP IN CREATING A SOFTWARE BUILD PLAN IS FOR THE SPM, STD, AND OTHER TECHNICAL LEADS WITHIN THE ORGANIZATION TO CREATE A BASIS OF ESTIMATE (BOE) FOR EACH CAPABILITY/FEATURE BEING DEVELOPED BASED ON ESTABLISHED SIZE AND PRODUCTIVITY METRICS (HOW MANY LINES OF CODE WILL BE DEVELOPED AND MANY HOURS IT WILL TAKE TO COMPLETE THE TASK).
   2) EACH BOE SHOULD ALSO HAVE A HIGH-LEVEL DESCRIPTION OF WHAT EACH CAPABILITY/FEATURE WILL ACHIEVE.

SW PROJECT PLANNING – CREATING THE SBP

3) SOFTWARE BUILD PLAN (SBP) STRUCTURE

- POINT OF DEPARTURE BUILD – TYPICALLY CONSIST OF CREATING A SOFTWARE BASELINE TO SUPPORT THIS PROGRAM FROM A PRODUCT LINE BASELINE. TYPICAL ACTIVITIES WOULD INCLUDE PORTING THE SOFTWARE AND ASSOCIATED TEST BED AND CREATING ADAPTATION DATA TO SUPPORT THE SPECIFIC PROGRAM.
- FRAMEWORK ENHANCEMENT BUILD – TYPICALLY CONSIST OF MODIFICATIONS TO THE PRODUCT LINE ARCHITECTURE TO SUPPORT THE NEW COMPONENTS OR FEATURES (EXAMPLES INCLUDE CORE FRAMEWORK ENHANCEMENTS, MESSAGING FRAMEWORK ENHANCEMENTS, AND PERFORMANCE ENHANCEMENTS).
- CAPABILITY/FEATURE BUILDS – ONE OR MORE BUILDS THAT ADD CAPABILITIES/FEATURES BASED ON THE CONCEPT OF MUST-HAVE FEATURES, NICE-TO-HAVE FEATURES, AND WOULD-BE-GREAT-TO-HAVE FEATURES IF POSSIBLE.

SW PROJECT PLANNING – CREATING THE SBP

4) SOFTWARE BUILD PLAN (SPB) CONTENT

BUILD 0 – POINT OF DEPARTURE BUILD

USE CASE 0.1 – DESCRIPTION OF CAPABILITY/FEATURE

USE CASE 0.2 – DESCRIPTION OF CAPABILITY/FEATURE

USE CASE 0.3 – DESCRIPTION OF CAPABILITY/FEATURE

BUILD 1 – FRAMEWORK ENHANCEMENT BUILD

USE CASE 1.1 – DESCRIPTION OF CAPABILITY/FEATURE

USE CASE 1.2 – DESCRIPTION OF CAPABILITY/FEATURE

USE CASE 1.3 – DESCRIPTION OF CAPABILITY/FEATURE

BUILD 2 – CAPABILITY/FEATURE A BUILD

USE CASE 2.1 – DESCRIPTION OF CAPABILITY/FEATURE

USE CASE 2.2 – DESCRIPTION OF CAPABILITY/FEATURE

USE CASE 2.3 – DESCRIPTION OF CAPABILITY/FEATURE

BUILD 3 – CAPABILITY/FEATURE B BUILD

USE CASE 3.1 – DESCRIPTION OF CAPABILITY/FEATURE

USE CASE 3.2 – DESCRIPTION OF CAPABILITY/FEATURE

USE CASE 3.3 – DESCRIPTION OF CAPABILITY/FEATURE

BUILD 4 – CAPABILITY/FEATURE C BUILD

USE CASE 4.1 – DESCRIPTION OF CAPABILITY/FEATURE

USE CASE 4.2 – DESCRIPTION OF CAPABILITY/FEATURE

USE CASE 4.3 – DESCRIPTION OF CAPABILITY/FEATURE

SW PROJECT PLANNING – DETAILED PLANNING

2. SUPPORT DETAILED IMS PLANNING

   ONCE YOUR SBP IS COMPLETED, YOU CAN BEGIN CREATING TECHNICAL AND SCHEDULE DEPENDENCIES BETWEEN THE DEVELOPMENTAL ACTIVITIES IN THE IMS.

   1) AS PART OF THE INITIAL PLANNING STAGE – ALL SOFTWARE ACTIVITIES MUST BE MAPPED TO A TASK WITHIN THE IMS AND HAVE HOURS AND BUDGET ASSOCIATED WITH THE TASK. ACTIVITIES WITHIN THE IMS ARE MODELED AS A

      - WORK PACKAGE – MEANING THE WORK DEFINED WITHIN THIS WORK PACKAGE IS AUTHORIZED TO BE WORKED ON TODAY OR WITHIN THE NEXT FEW MONTHS.
      - PLANNING PACKING – MEANING THE WORK DEFINED WITHIN THIS WORK PACKAGE WILL BE AUTHORIZED TO BE PERFORMED SOMETIME IN THE FUTURE.
      - LEVEL-OF-EFFORT – MEANING CREDIT FOR THE WORK AUTOMATICALLY OCCURS WHETHER OR NOT ANYTHING GETS ACCOMPLISHED OR AN INDIVIDUAL WORKS A TASK (E.G. SPM OR SUPPORT LABOR)

SW PROJECT PLANNING – DETAILED PLANNING

2) MODELING EACH CATEGORY WITHIN THE SOFTWARE BID IN THE IMS

- SOFTWARE MANAGEMENT AND SUPPORT ACTIVITIES – ARE PLANNED IN THE IMS AS EITHER A WORK/PLANNING PACKAGE TASK OR A LOE TASK AND LINKED SEQUENTIALLY OVER A PERIOD OF PERFORMANCE.
- SOFTWARE DEVELOPMENT ACTIVITIES – ARE PLANNED IN THE IMS AS EITHER A WORK/PLANNING PACKAGE DISCRETE TASK AND LINKED BASED ON THEIR TECHNICAL AND SCHEDULE DEPENDENCIES TO OTHER DEVELOPMENTAL DISCRETE TASKS.

  NOTE: IMS LOGIC MUST TAKE THE SOFTWARE DEVELOPMENT MODEL SPECIFIED WITHIN THE SOFTWARE DEVELOPMENT PLAN (SDP) INTO ACCOUNT. FOR EXAMPLE, IF THE SDP STATES THAT AN ITERATIVE APPROACH FOR SW DEVELOPMENT IS BEING USED THEN THE IMS LOGIC MUST REFLECT THAT MODEL AS PART OF THE DEVELOPMENT SCHEDULE.
- SOFTWARE TEST ACTIVITIES – ARE PLANNED IN THE IMS AS EITHER A WORK/PLANNING PACKAGE DISCRETE TASK AND LINKED BASED ON THEIR TECHNICAL AND SCHEDULE DEPENDENCIES TO DEVELOPMENTAL DISCRETE TASKS.
- SOFTWARE MAINTENANCE ACTIVITIES – ARE PLANNED IN THE IMS AS EITHER A WORK/PLANNING PACKAGE DISCRETE TASK AND LINKED BASED ON THEIR TECHNICAL AND SCHEDULE DEPENDENCIES TO DEVELOPMENTAL AND TEST DISCRETE TASKS.

THE SUM OF ALL TASKS IN THE BASELINED IMS WILL TRANSLATE TO THE BASELINE SCHEDULE (IN HOURS) AND BUDGET (IN DOLLARS) TO COMPLETE THE WORK.

MY RECOMMENDATION IS TO NEVER USE AN LOE TASK IN THE IMS UNLESS YOU ARE FORCED TO DO SO!

SW PROJECT PLANNING – ESTABLISHING YOUR COST BASELINE

3. JUSTIFY YOUR ORGANIZATIONS BUDGET TO PROGRAM MANAGEMENT.

   1) EACH TASK WILL NEED TO HAVE A BASIS OF ESTIMATE (BOE) BASED ON KNOWN METRICS.   ONCE PROGRAM MANAGEMENT AGREES TO THE BUDGET YOUR TEAM WILL BE ALLOCATED, HOURS TO COMPLETE EACH TASK ALONG WITH THE CORRESPONDING BUDGET TO COMPLETE EACH TASK WILL BE ASSIGNED TO EACH TASK IN THE IMS.

   2) REMEMBER FOR A PROGRAM TO BE SUCCESSFUL, YOU NEED TO EFFECTIVELY MANAGE COST BY USING A SIMILAR BUFFER STRATEGY AS WHEN YOU WERE DEALING WITH THE SCHEDULE.

      - EFFECTIVE PROGRAMS WILL CREATE A BUDGET RESERVE BY ADDING PRODUCTIVITY CHALLENGES TO EACH ORGANIZATION.
      - AS THE SPM, YOU WILL NEED TO DETERMINE IF YOU ARE GOING TO LEVEL THAT PRODUCTIVITY CHALLENGE EQUALLY ACROSS ALL DEVELOPMENT EFFORTS, SUPPORT, AND MANAGEMENT ACTIVITIES OR ONLY ON A SUBSET OF THE DEVELOPMENT EFFORT.

      EVEN IF YOU ARE NOT FORCED TO TAKE A PRODUCTIVITY CHALLENGE, IT IS ALWAYS A GOOD IDEA TO CREATE AN INTERNAL BUDGET RESERVE THAT YOU CAN MANAGE SINCE THE TEAM THAT IMPLEMENTS A CAPABILITY/FEATURE ULTIMATELY NEEDS TO BUY INTO THE BUDGET THEY HAVE BEEN ALLOCATED.

SW PROJECT PLANNING – ACCOUNTING FOR RISKS

3) ACCOUNT FOR RISK IN YOUR COST BASELINE

IN YOU ARE FORCED TO TAKE A PROGRAM PRODUCTIVITY CHALLENGE ALWAYS ADD A PROGRAM RISK ASSOCIATED WITH THE PRODUCTIVITY CHALLENGE.

ANOTHER TYPICAL SOFTWARE RISK THAT SHOULD ALWAYS BE INCLUDED IS A SW SIZE RISK IF NEW TECHNOLOGY WILL BE USED AND YOUR METRICS MANY NOT ACCURATELY REFLECT THE EFFORT TO DEVELOP THE PRODUCT.

ANOTHER TYPE OF SOFTWARE RISK THAT SHOULD ALWAYS BE INCLUDED IS A RISK DEALING WITH CAPABILITIES/FEATURES THAT HAVE A PERFORMANCE REQUIREMENT ASSOCIATED WITH THEM THAT YOUR SYSTEM CURRENTLY DOES NOT MEET.

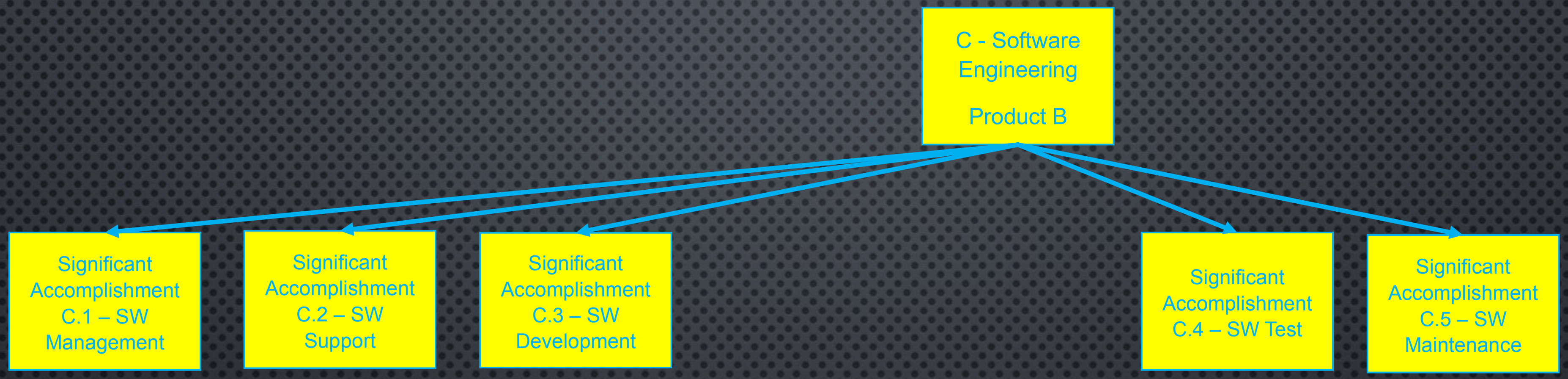
FINALLY, YOU SHOULD INCLUDE ANY RISKS DUE TO OBSOLESCENCE, HARDWARE AVAILABILITY, AND SUB-CONTRACTOR ISSUES.

IF THERE ARE HIGH RISK ITEMS THAT ARE A PART OF YOUR BID, YOU MAY WANT TO CONSIDER ASKING THE PROGRAM TO FUND A RISK MITIGATION ACTIVITY AS PART OF YOUR INITIAL SOFTWARE BASELINE.

SW PROJECT PLANNING – DECOMPOSING THE WBS

4. AS PREVIOUSLY STATED, THE HIGH-LEVEL WBS IS TYPICALLY DEFINED AS PART OF THE PROPOSAL EFFORT IN RESPONSE TO A CUSTOMER RFP.
   - YOUR GOAL IS TO CREATE THE LOWER LEVEL WBS STRUCTURE SO THAT YOU CAN ACCURATELY MANAGE THE HOURS AND ASSOCIATED COST TO COMPLETE EACH TASK YOU ARE MANAGING.
   - THIS DATA WILL NOT ONLY HELP YOU MANAGE THIS PROGRAM BUT WILL PROVIDE YOUR ORGANIZATION WITH DATA THEY CAN USE WHEN BIDDING SW PRODUCTIVITY ON FUTURE PROGRAMS.
   - ALTHOUGH YOU ARE ULTIMATELY RESPONSIBLE FOR THE COST TO COMPLETE THE WORK YOU HAVE SIGNED UP FOR MY RECOMMENDATION IS TO ALWAYS MANAGE YOUR PROGRESS IN HOURS VS. DOLLARS. COMPLETING A TASK ON OR IN LESS HOURS THAN YOUR BASELINE HOURS WILL TELL YOU HOW WELL YOUR TEAM IS PERFORMING.
   - THE COST OF PERFORMING THE WORK IS DEPENDENT ON OTHER FINANCIAL FACTORS THAT MAY BE OUTSIDE OF YOUR CONTROL. BY UNDERSTANDING YOUR TEAM’S PERFORMANCE BASED ON HOURS, YOU WILL BE ABLE TO EXPLAIN YOUR TEAM’S PERFORMANCE BASED ON DOLLARS ONCE YOU GAIN AN UNDERSTANDING OF THOSE OUTSIDE INFLUENCES.

SW PROJECT PLANNING – DECOMPOSING THE WBS



C - Software Engineering

Product B

Significant Accomplishment C.1 – SW Management

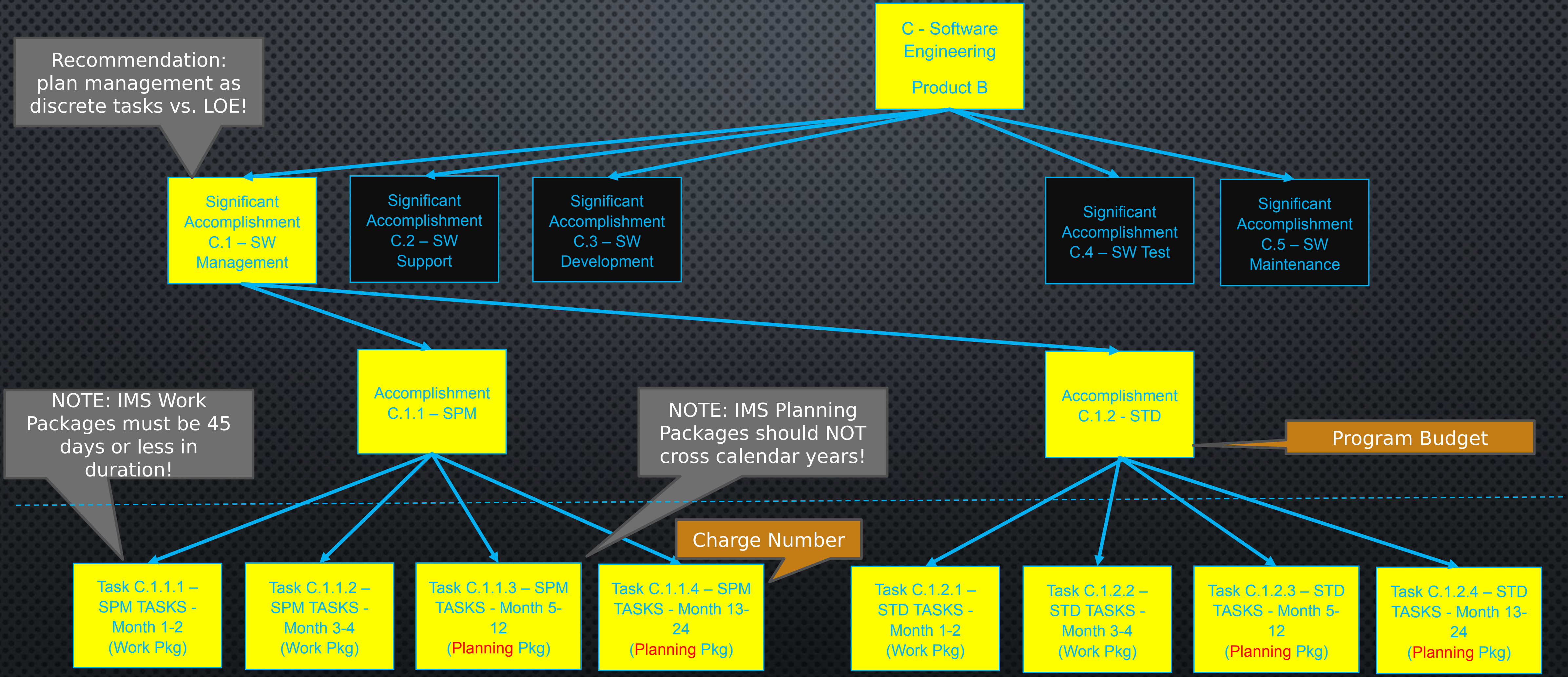
Significant Accomplishment C.2 – SW Support

Significant Accomplishment C.3 – SW Development

Significant Accomplishment C.4 – SW Test

Significant Accomplishment C.5 – SW Maintenance

SW PROJECT PLANNING – DECOMPOSING THE WBS



C - Software Engineering

Product B

Recommendation: plan management as discrete tasks vs. LOE!

Significant Accomplishment C.1 – SW Management

Significant Accomplishment C.2 – SW Support

Significant Accomplishment C.3 – SW Development

Significant Accomplishment C.4 – SW Test

Significant Accomplishment C.5 – SW Maintenance

Accomplishment C.1.1 – SPM

Accomplishment C.1.2 - STD

Program Budget

NOTE: IMS Work Packages must be 45 days or less in duration!

NOTE: IMS Planning Packages should NOT cross calendar years!

Charge Number

Task C.1.1.1 – SPM TASKS - Month 1-2 (Work Pkg)

Task C.1.1.2 – SPM TASKS - Month 3-4 (Work Pkg)

Task C.1.1.3 – SPM TASKS - Month 5-12 (Planning Pkg)

Task C.1.1.4 – SPM TASKS - Month 13-24 (Planning Pkg)

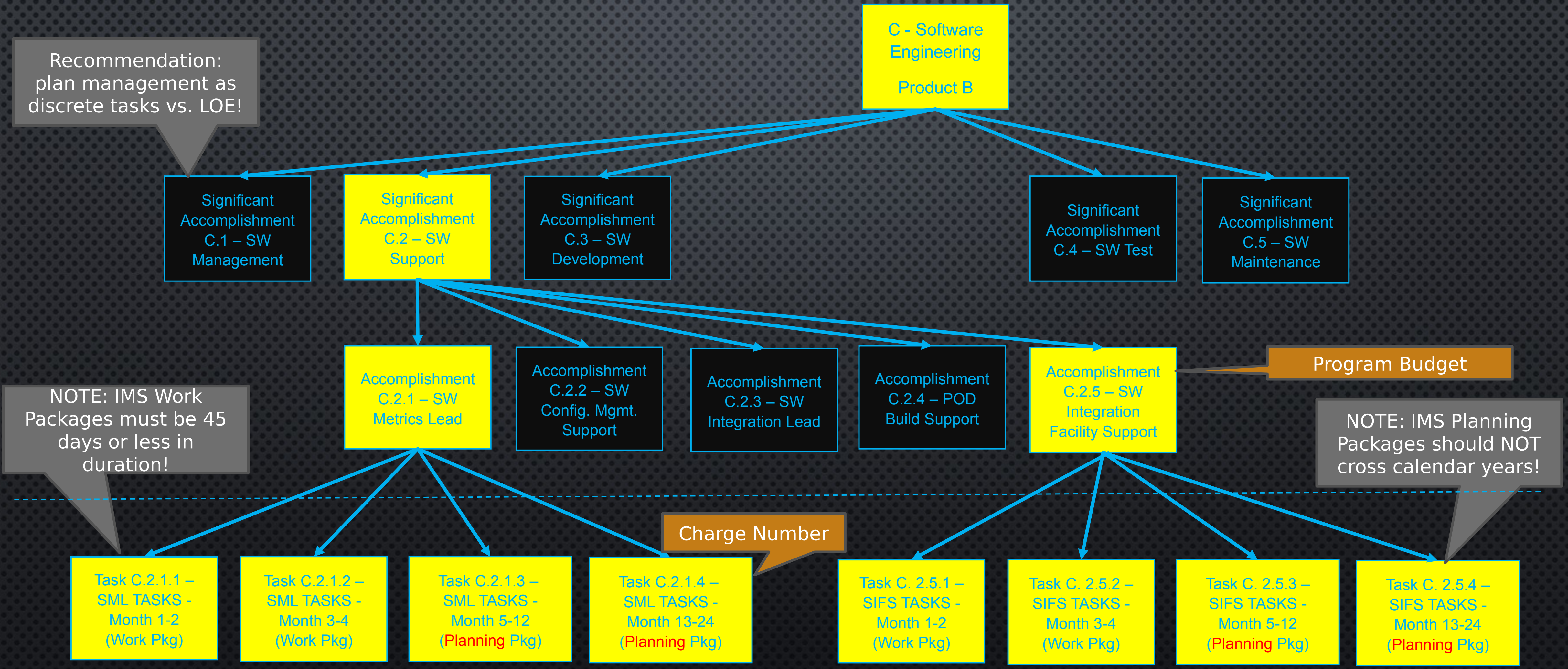
Task C.1.2.1 – STD TASKS - Month 1-2 (Work Pkg)

Task C.1.2.2 – STD TASKS - Month 3-4 (Work Pkg)

Task C.1.2.3 – STD TASKS - Month 5-12 (Planning Pkg)

Task C.1.2.4 – STD TASKS - Month 13-24 (Planning Pkg)

SW PROJECT PLANNING – DECOMPOSING THE WBS



C - Software Engineering

Product B

Significant Accomplishment C.1 – SW Management

Significant Accomplishment C.2 – SW Support

Significant Accomplishment C.3 – SW Development

Significant Accomplishment C.4 – SW Test

Significant Accomplishment C.5 – SW Maintenance

Accomplishment C.2.1 – SW Metrics Lead

Accomplishment C.2.2 – SW Config. Mgmt. Support

Accomplishment C.2.3 – SW Integration Lead

Accomplishment C.2.4 – POD Build Support

Accomplishment C.2.5 – SW Integration Facility Support

Task C.2.1.1 – SML TASKS - Month 1-2 (Work Pkg)

Task C.2.1.2 – SML TASKS - Month 3-4 (Work Pkg)

Task C.2.1.3 – SML TASKS - Month 5-12 (Planning Pkg)

Task C.2.1.4 – SML TASKS - Month 13-24 (Planning Pkg)

Task C. 2.5.1 – SIFS TASKS - Month 1-2 (Work Pkg)

Task C. 2.5.2 – SIFS TASKS - Month 3-4 (Work Pkg)

Task C. 2.5.3 – SIFS TASKS - Month 5-12 (Planning Pkg)

Task C. 2.5.4 – SIFS TASKS - Month 13-24 (Planning Pkg)

Recommendation: plan management as discrete tasks vs. LOE!

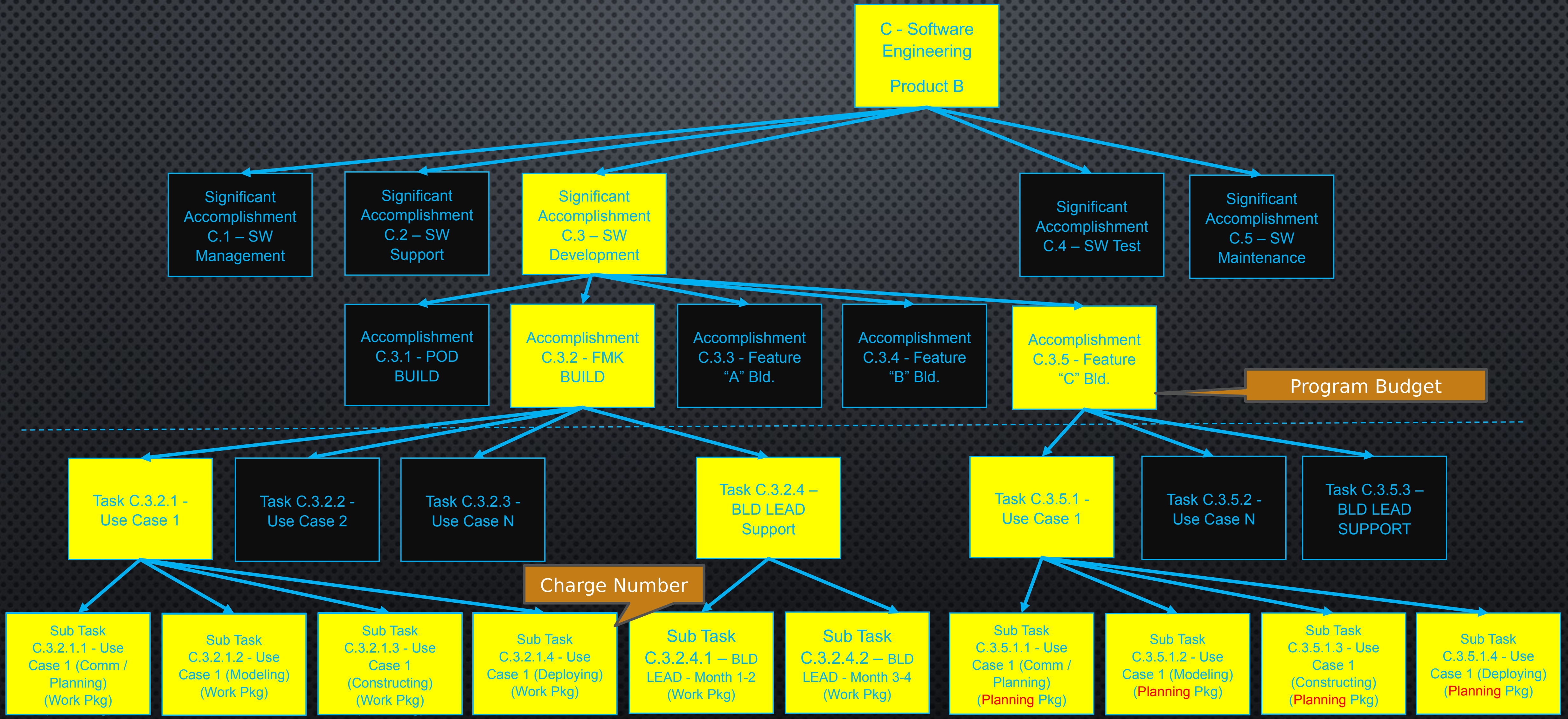
NOTE: IMS Work Packages must be 45 days or less in duration!

Program Budget

Charge Number

NOTE: IMS Planning Packages should NOT cross calendar years!

SW PROJECT PLANNING – DECOMPOSING THE WBS



C - Software Engineering

Product B

Significant Accomplishment C.1 – SW Management

Significant Accomplishment C.2 – SW Support

Significant Accomplishment C.3 – SW Development

Significant Accomplishment C.4 – SW Test

Significant Accomplishment C.5 – SW Maintenance

Accomplishment C.3.1 - POD BUILD

Accomplishment C.3.2 - FMK BUILD

Accomplishment C.3.3 - Feature “A” Bld.

Accomplishment C.3.4 - Feature “B” Bld.

Accomplishment C.3.5 - Feature “C” Bld.

Program Budget

Task C.3.2.1 - Use Case 1

Task C.3.2.2 - Use Case 2

Task C.3.2.3 - Use Case N

Task C.3.2.4 – BLD LEAD Support

Task C.3.5.1 - Use Case 1

Task C.3.5.2 - Use Case N

Task C.3.5.3 – BLD LEAD SUPPORT

Charge Number

Sub Task C.3.2.1.1 - Use Case 1 (Comm / Planning) (Work Pkg)

Sub Task C.3.2.1.2 - Use Case 1 (Modeling) (Work Pkg)

Sub Task C.3.2.1.3 - Use Case 1 (Constructing) (Work Pkg)

Sub Task C.3.2.1.4 - Use Case 1 (Deploying) (Work Pkg)

Sub Task C.3.2.4.1 – BLD LEAD - Month 1-2 (Work Pkg)

Sub Task C.3.2.4.2 – BLD LEAD - Month 3-4 (Work Pkg)

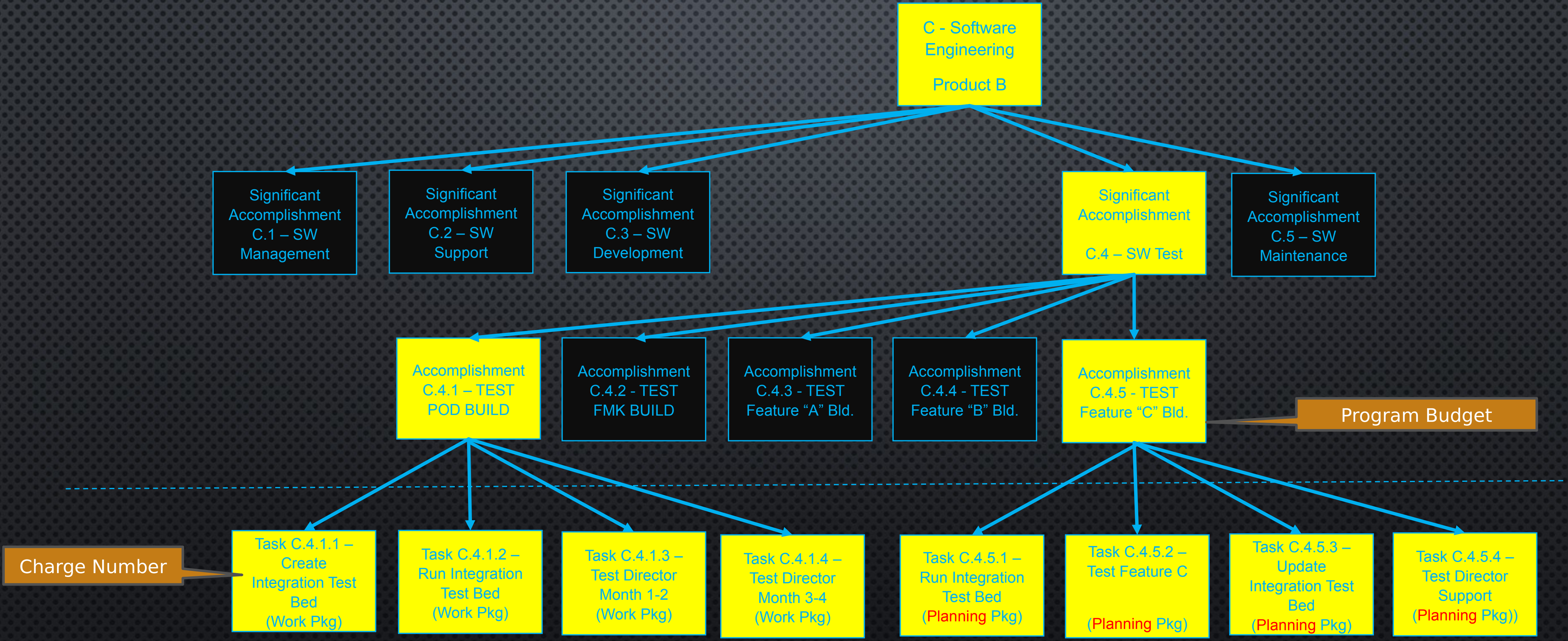
Sub Task C.3.5.1.1 - Use Case 1 (Comm / Planning) (Planning Pkg)

Sub Task C.3.5.1.2 - Use Case 1 (Modeling) (Planning Pkg)

Sub Task C.3.5.1.3 - Use Case 1 (Constructing) (Planning Pkg)

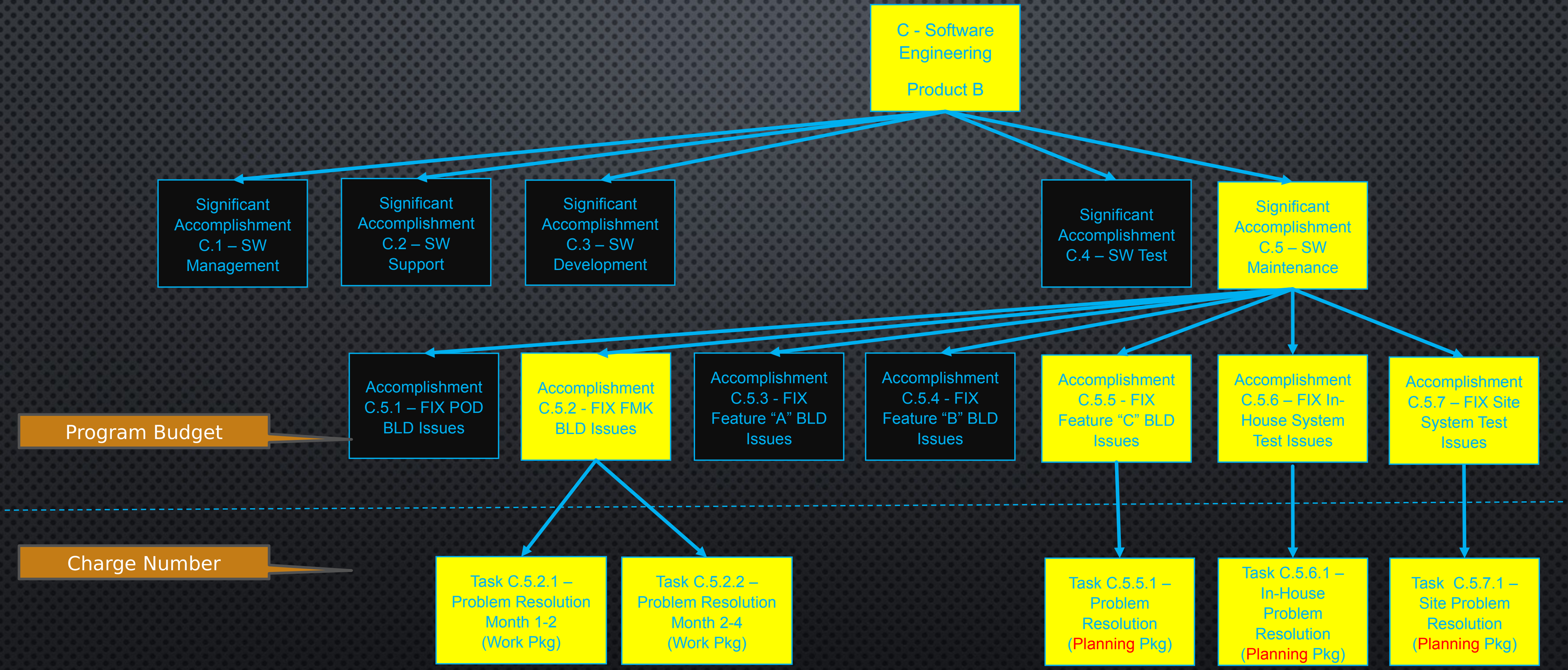
Sub Task C.3.5.1.4 - Use Case 1 (Deploying) (Planning Pkg)

SW PROJECT PLANNING – DECOMPOSING THE WBS



C - Software Engineering
Product B
Significant Accomplishment C.1 – SW Management
Significant Accomplishment C.2 – SW Support
Significant Accomplishment C.3 – SW Development
Significant Accomplishment C.4 – SW Test
Significant Accomplishment C.5 – SW Maintenance
Accomplishment C.4.1 – TEST POD BUILD
Accomplishment C.4.2 - TEST FMK BUILD
Accomplishment C.4.3 - TEST Feature “A” Bld.
Accomplishment C.4.4 - TEST Feature “B” Bld.
Accomplishment C.4.5 - TEST Feature “C” Bld.
Program Budget
Charge Number
Task C.4.1.1 – Create Integration Test Bed (Work Pkg)
Task C.4.1.2 – Run Integration Test Bed (Work Pkg)
Task C.4.1.3 – Test Director Month 1-2 (Work Pkg)
Task C.4.1.4 – Test Director Month 3-4 (Work Pkg)
Task C.4.5.1 – Run Integration Test Bed (Planning Pkg)
Task C.4.5.2 – Test Feature C (Planning Pkg)
Task C.4.5.3 – Update Integration Test Bed (Planning Pkg)
Task C.4.5.4 – Test Director Support (Planning Pkg))

SW PROJECT PLANNING – DECOMPOSING THE WBS



C - Software Engineering

Product B

Significant Accomplishment C.1 – SW Management

Significant Accomplishment C.2 – SW Support

Significant Accomplishment C.3 – SW Development

Significant Accomplishment C.4 – SW Test

Significant Accomplishment C.5 – SW Maintenance

Program Budget

Accomplishment C.5.1 – FIX POD BLD Issues

Accomplishment C.5.2 - FIX FMK BLD Issues

Accomplishment C.5.3 - FIX Feature “A” BLD Issues

Accomplishment C.5.4 - FIX Feature “B” BLD Issues

Accomplishment C.5.5 - FIX Feature “C” BLD Issues

Accomplishment C.5.6 – FIX In-House System Test Issues

Accomplishment C.5.7 – FIX Site System Test Issues

Charge Number

Task C.5.2.1 – Problem Resolution Month 1-2 (Work Pkg)

Task C.5.2.2 – Problem Resolution Month 2-4 (Work Pkg)

Task C.5.5.1 – Problem Resolution (Planning Pkg)

Task C.5.6.1 – In-House Problem Resolution (Planning Pkg)

Task C.5.7.1 – Site Problem Resolution (Planning Pkg)

SW PROJECT PLANNING – SDP

5. CREATING A SOFTWARE DEVELOPMENT PLAN (SDP)

   ASIDE FROM DEVELOPING THE SBP, THE SDP IS THE REMAINING DOCUMENT THAT IS REQUIRED AS PART OF THE INITIAL SOFTWARE DEVELOPMENT PHASE.

   THE SDP CAPTURES THE MANAGEMENT APPROACH AND ENGINEERING ENVIRONMENT EFFORT ASSOCIATED WITH THE PROGRAM. IT IDENTIFIES:

   - THE SOFTWARE ORGANIZATIONAL STRUCTURE WITH RESPECT TO THE PROGRAM,
   - THE HIGH-LEVEL SOFTWARE SCHEDULE WITH KEY MILESTONES,
   - THE SOFTWARE RISKS ASSOCIATED WITH THE PROPOSED DEVELOPMENT,
   - THE SOFTWARE MEASUREMENT AND ANALYSIS PLAN (TAKING NEW, MODIFIED, AND REUSE DEVELOPMENT INTO ACCOUNT),
   - THE SOFTWARE CONFIGURATION MANAGEMENT PLAN (ENSURES ONLY TESTED SOFTWARE GETS CHECKED ONTO A BUILD),
   - THE SOFTWARE ENGINEERING ENVIRONMENT (IN-HOUSE LAB FACILITIES),

   THE SDP ALSO CAPTURES THE TECHNICAL APPROACH TO DEVELOPING THE SOFTWARE ASSOCIATED WITH A PROGRAM.

   - THE SOFTWARE DEVELOPMENT PARADIGM THAT WILL BE FOLLOWED INCLUDING ANY STANDARDS,
   - THE SOFTWARE PROCESSES THAT WILL BE FOLLOWED (THESE CAN BE SEPARATE DOCUMENTS THAT ARE REFERENCED) AND ARTIFACTS THAT WILL BE PRODUCED,
   - THE USE AND INTEGRATION OF FREE AND OPEN-SOURCE SOFTWARE (FOSS) AND COMMERCIAL OFF-THE-SHELF (COTS) SOFTWARE,
   - THE SOFTWARE BUILD PLAN (TYPICALLY SPECIFIED IN A SEPARATE DOCUMENT).

SW PROJECT PLANNING – ORGANIZATION

PROGRAM ORGANIZATIONAL STRUCTURED

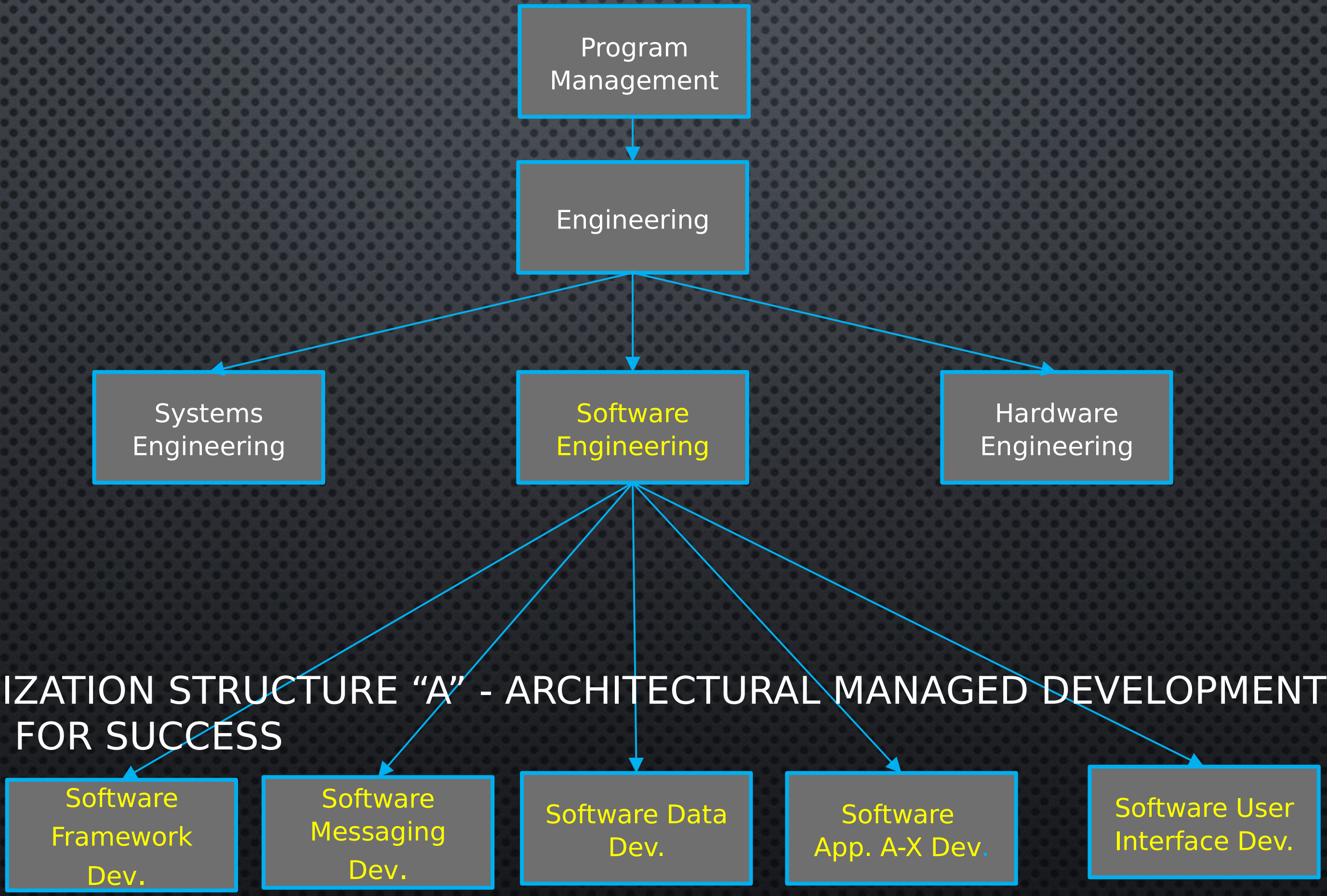
THE ORGANIZATIONAL STRUCTURE OF THE PROGRAM IS TYPICALLY DETERMINED AT THE TIME THE PROGRAM IS BID SINCE IT NEEDS TO ACCOUNT FOR THE USE OF AN INTEGRATED PRODUCT TEAM (IPT) DEVELOPMENT STRUCTURE VS. A NON-IPT STRUCTURE.

PROGRAM STRUCTURE HAS A DIRECT IMPACT ON YOUR ORGANIZATION’S ABILITY TO IMPLEMENT THE ITERATIVE OR AGILE APPROACH SPECIFIED IN THE SDP DOCUMENT.

FOR EXAMPLE, A PROGRAM STRUCTURE THAT IS ORGANIZATIONAL OR ARCHITECTURAL BASED VS. USING AN IPT STRUCTURE IS MORE LIKELY TO HAVE LIMITATIONS IN RECEIVING ALL OF THE BENEFITS OF THE ITERATIVE OR AGILE APPROACH THE SW TEAM IS IMPLEMENTING AS SHOWN IN THE DIAGRAMS BELOW.

PROGRAM ORGANIZATION STRUCTURE “A”

DISCIPLINE BASED ORGANIZATIONAL STRUCTURE - PROGRAMS AND DISCIPLINES HAVE THEIR OWN GOALS FOR SUCCESS

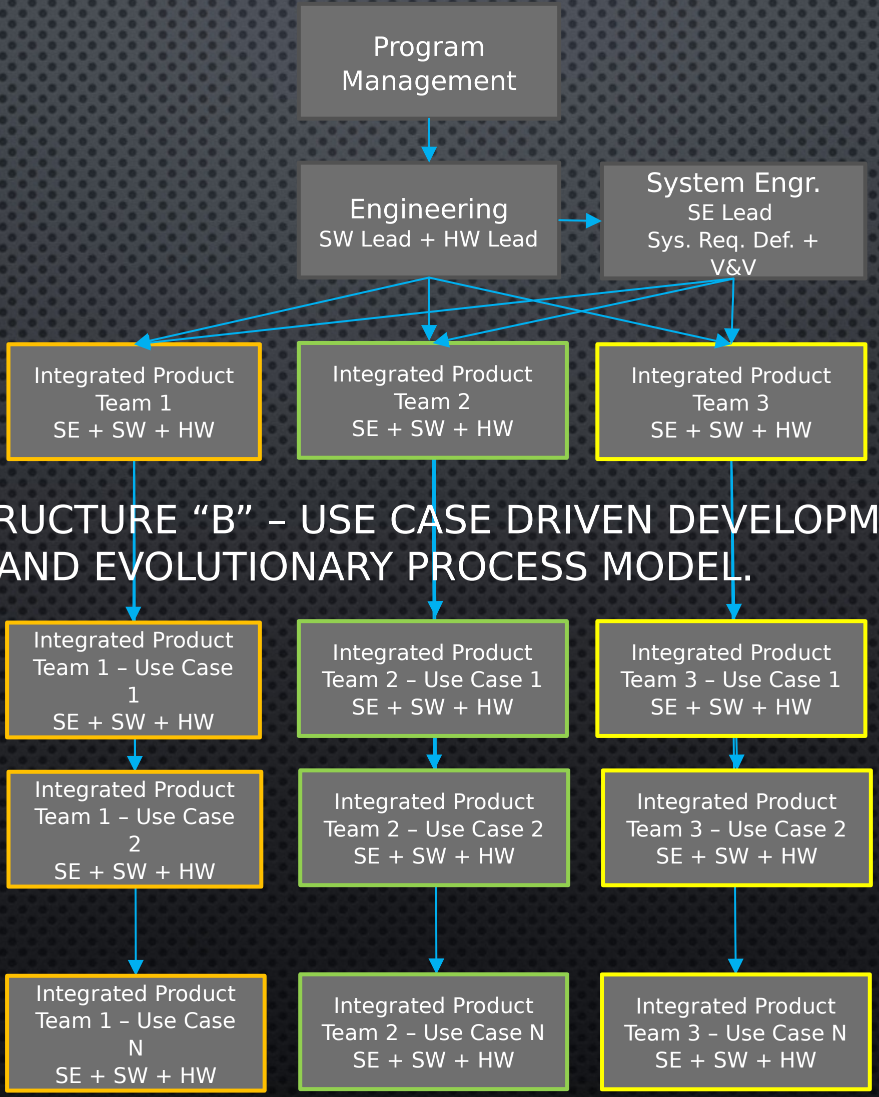


SOFTWARE ORGANIZATION STRUCTURE “A” - ARCHITECTURAL MANAGED DEVELOPMENT TEAMS HAVE THEIR OWN GOALS FOR SUCCESS

PROGRAM ORGANIZATION STRUCTURE "B"

INTEGRATED PRODUCT TEAM ORGANIZATIONAL STRUCTURE – BALANCES PROGRAMS AND GOALS FOR SUCCESS

SOFTWARE ORGANIZATION STRUCTURE "B" – USE CASE DRIVEN DEVELOPMENT TEAMS CAN MORE EASILY FOLLOW AN ITERATIVE AND EVOLUTIONARY PROCESS MODEL.


Program Management
Engineering
SW Lead + HW Lead
System Engr.
SE Lead
Sys. Req. Def. + V&V
Integrated Product Team 1
SE + SW + HW
Integrated Product Team 2
SE + SW + HW
Integrated Product Team 3
SE + SW + HW
Integrated Product Team 1 – Use Case 1
SE + SW + HW
Integrated Product Team 2 – Use Case 1
SE + SW + HW
Integrated Product Team 3 – Use Case 1
SE + SW + HW
Integrated Product Team 1 – Use Case 2
SE + SW + HW
Integrated Product Team 2 – Use Case 2
SE + SW + HW
Integrated Product Team 3 – Use Case 2
SE + SW + HW
Integrated Product Team 1 – Use Case N
SE + SW + HW
Integrated Product Team 2 – Use Case N
SE + SW + HW
Integrated Product Team 3 – Use Case N
SE + SW + HW

SW PROJECT PLANNING – CONFIGURATION MGMT.

PROGRAM SW CONFIGURATION MANAGEMENT PLAN

THE SW CONFIGURATION MANAGEMENT PLAN DOCUMENTS HOW SOFTWARE ARTIFACTS ARE CONTROLLED AND HOW CHANGE TO THESE ARTIFACTS IS MANAGED THROUGHOUT THE EXECUTION OF THE PROGRAM. THIS INCLUDES REQUIREMENTS TRACEABILITY, DESIGN DOCUMENTATION, CODE, TEST DOCUMENTATION, SOFTWARE PROCESS DOCUMENTS, ETC.

THE SW CONFIGURATION MANAGEMENT PLAN MAY ALSO SPECIFY SPECIFIC TOOLS USED TO CONTROL THE CONFIGURATION OF THE REQUIREMENTS AND CODE IN PLANT AS WELL AS AT THE CUSTOMER FACILITY.

SW PROJECT PLANNING – INTEGRATION FACILITY

PROGRAM SW INTEGRATION AND TEST ENVIRONMENT

THE ONLY WAY TO ENSURE A PRODUCT WORKS AS SPECIFIED IS TO TEST IT IN ACTUAL ENVIRONMENT IN WHICH IT WILL BE USED. UNFORTUNATE SOFTWARE ERRORS FOUND IN THE ACTUAL DELIVERABLE ENVIRONMENT ARE THE MOST EXPENSIVE TYPES OF ERRORS TO FIX.

THE SOONER A SW ERROR CAN BE DETECTED AND FIX DIRECTLY CORRELATES TO COST ASSOCIATED WITH FIXING THE ERROR.

- ERRORS DISCOVERED AND FIXED DURING UNIT TEST ARE THE CHEAPEST ERROR TO FIX.
- ERRORS DISCOVERED DURING INTEGRATION/SYSTEM TESTING THAT ARE FOUND/FIXED IN THE INTEGRATION AND TEST ENVIRONMENT END UP BEING MORE EXPENSIVE TO FIX SINCE THEY IMPACTS A LARGER NUMBER OF INDIVIDUALS AND REQUIRES A NUMBER OF TEST ASSETS IN THE INTEGRATION FACILITY TO VERIFY THE ISSUE WAS FIXED. HOWEVER, THE COST OF FIXING A SOFTWARE ERROR PRIOR TO THE PRODUCT LEAVING THE SITE IS KEY TO MINIMIZING PROGRAM COSTS FOR PROBLEM RESOLUTION.

SW PROJECT PLANNING – INTEGRATION FACILITY

HENCE IT IS CRITICAL THAT A PLAN IS PUT IN PLACE DURING PROGRAM STARTUP THAT CREATES AN INTEGRATION TEST ENVIRONMENT THAT ACCURATELY REFLECTS THE TARGET ENVIRONMENT AND HAS ENOUGH EQUIPMENT TO SUPPORT THE DEVELOPMENT SCHEDULE DEFINED BY THE PROGRAM.

- HARDWARE COMPONENTS THAT NEED TO BE CONSIDERED INCLUDED: PHYSICAL SPACE, NUMBER OF WORKSTATIONS AND SERVERS THAT COMPRISE A STRING, NUMBER OF STRINGS THAT COMPRISE A SITE, NUMBER OF SITES AT THE TARGET DOMAIN.
- SCHEDULING COMPONENTS THAT NEED TO BE CONSIDERED INCLUDES AVERAGE AND PEEK UTILIZATION OF THE EQUIPMENT.

SW PROJECT PLANNING – SW PROCESSES

SOFTWARE PROCESS THAT WILL BE FOLLOWED DURING THE EXECUTION OF THE PROGRAM CAN EITHER BE DIRECTLY CONTAINED IN THE SDP OR DEFINED IN SEPARATE PROGRAM DOCUMENTS.

EXAMPLES OF SOFTWARE PROCESS THAT WOULD BE REFERENCED WITHIN THE SDP ARE HIGHLIGHTED IN THE DIAGRAM TO THE RIGHT.

NOTE: SOMETIMES IT IS CHEAPER TO INCLUDE A COTS PRODUCT AS PART OF THE SOLUTION TO THE SYSTEM BEING DESIGN OR TO USE A FOSS PRODUCT. IF THAT SITUATION, IT IS IMPORTANT TO ADDRESS HOW THE COTS/FOSS PRODUCT WILL BE INTEGRATED, TESTED, AND CONTROLLED AS PART OF THE SOFTWARE DEVELOPMENT PLAN.

SOFTWARE ENGINEERING DISCIPLINE

SOFTWARE ORGANIZATION

KEY PROGRAM PROCESSES:

1. SOFTWARE DEVELOPMENT PLAN (SDP),
2. SOFTWARE BUILD PLAN (SBP),
3. SOFTWARE PRELIMINARY AND DETAIL DESIGN,
4. SOFTWARE CODE & UNIT TEST PLAN,
5. SOFTWARE CODING STANDARDS (LANGUAGE SPECIFIC),
6. SOFTWARE INTEGRATION PLAN,
7. SOFTWARE VERIFICATION PLAN,
8. SOFTWARE CONFIGURATION MANAGEMENT PLAN

REFERENCES

AGILE & ITERATIVE DEVELOPMENT, A MANAGER'S GUIDE, CRAIG LARMAN, EIGHTH EDITION, ADDISON WESLEY, NEW YORK, NY, COPYRIGHT 2004 BY PEARSON EDUCATION , INC.

MITRE SYSTEMS ENGINEERING GUIDE *HTTPS://WWW.MITRE.ORG/PUBLICATIONS/SYSTEMS-ENGINEERING-GUIDE/ACQUISITION-SYSTEMS-ENGINEERING/ACQUISITION-PROGRAM-PLANNING/INTEGRATED-MASTER-SCHEDULE-IMSINTEGRATED-MASTER-PLAN-IMP-APPLICATION*

PRINCIPLES OF SOFTWARE ENGINEERING MANAGEMENT, BY TOM GILB, COPYRIGHT 1988 BY ADDISON-WESLEY PUBLISHING COMPANY, NEW YORK, NY

SOFTWARE ENGINEERING, A PRACTITIONER'S APPROACH, EIGHTH EDITION, BY ROGER S. PRESSMAN AND BRUCE R. MAXIM, COPYRIGHT 2015 BY MCGRAW HILL, NEW YORK, NY

PROCESS

AGILE & ITERATIVE DEVELOPMENT (CHAPTER 2)

A MANAGER'S GUIDE BY: CRAIG LARMAN

WEEK 3: ITERATIVE & EVOLUTIONARY METHODS

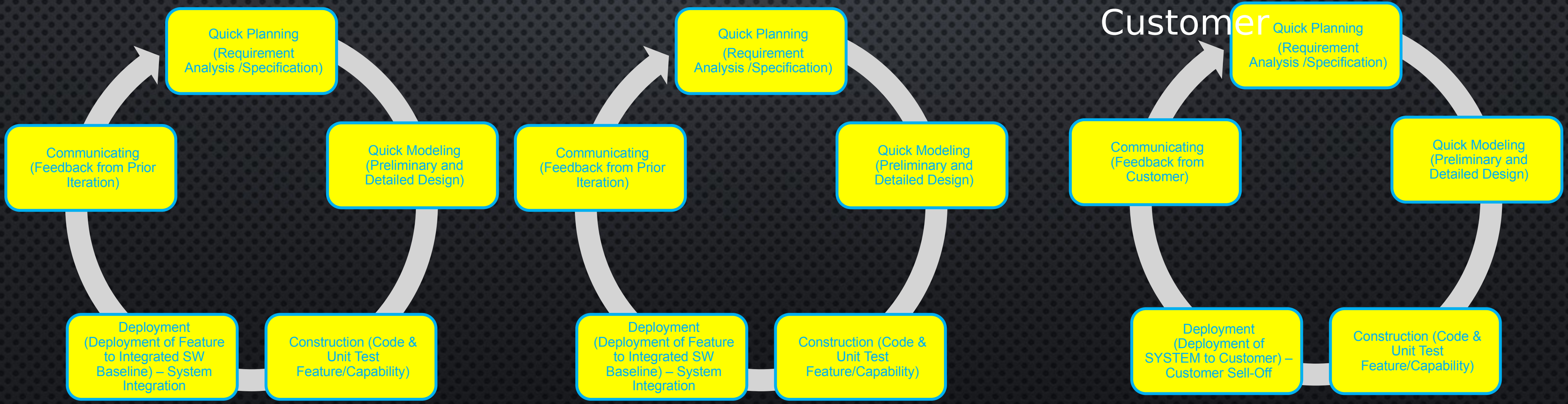
BY: JOSEPH MARTINAZZI

ITERATIVE (INCREMENTAL) DEVELOPMENT

ITERATIVE DEVELOPMENT LIFECYCLE MODEL

- **AN ITERATIVE APPROACH TO DEVELOPING SOFTWARE IS WHERE REQUIREMENT ANALYSIS/SPECIFICATION, DESIGN, CODE & TEST OCCUR IN MULTIPLE ITERATIONS OVER THE LIFECYCLE OF THE PROGRAM.**

Build 1 – FMK → Build 2 – Capabilities → Build 3 – Capabilities → Build X – Release to Customer



Quick Planning (Requirement Analysis /Specification)

Quick Modeling (Preliminary and Detailed Design)

Construction (Code & Unit Test Feature/Capability)

Deployment (Deployment of Feature to Integrated SW Baseline) – System Integration

Communicating (Feedback from Prior Iteration)

Quick Planning (Requirement Analysis /Specification)

Quick Modeling (Preliminary and Detailed Design)

Construction (Code & Unit Test Feature/Capability)

Deployment (Deployment of Feature to Integrated SW Baseline) – System Integration

Communicating (Feedback from Prior Iteration)

Quick Planning (Requirement Analysis /Specification)

Quick Modeling (Preliminary and Detailed Design)

Construction (Code & Unit Test Feature/Capability)

Deployment (Deployment of SYSTEM to Customer) – Customer Sell-Off

Communicating (Feedback from Customer)

System Capabilities grow incrementally

ITERATIVE (INCREMENTAL) DEVELOPMENT

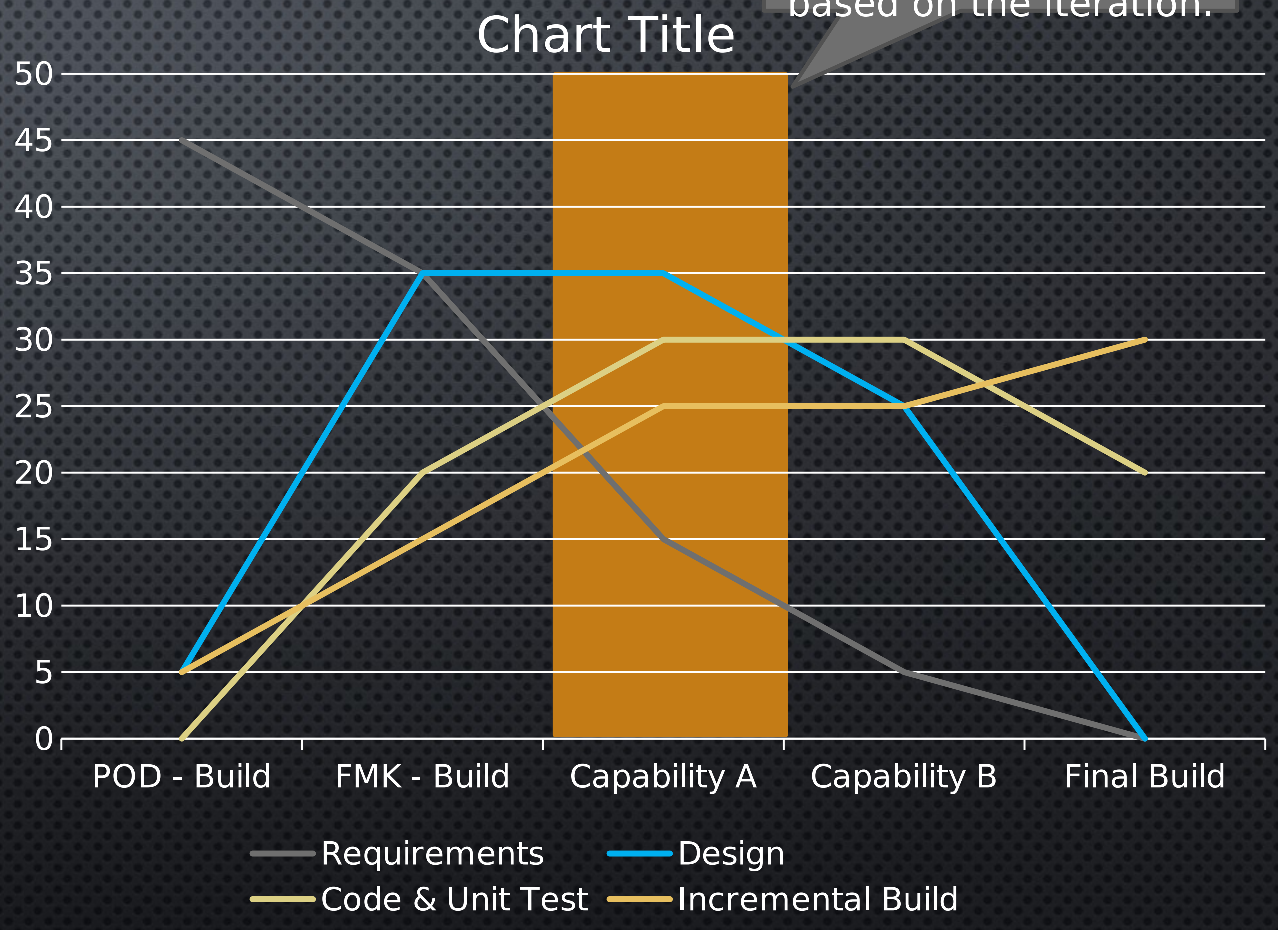
ITERATIVE DEVELOPMENT LIFECYCLE MODEL

- **THE GOAL OF EACH ITERATION IS TO DELIVER TESTED CODE CONTAINING CAPABILITIES/FEATURES THAT ARE BUILT UP SEQUENTIALLY IN AN INTEGRATED SOFTWARE BASELINE OVER TIME.**
- ***THIS APPROACH ENABLES:***
  - ***MULTIPLE TEAMS OF SOFTWARE DEVELOPERS TO CONTRIBUTE TESTED CODE TO AN INTEGRATED AND PARTIALLY TESTED STABLE BASELINE.***
  - ***RUN REGRESSION TEST AGAINST THE BASELINE TO ENSURE BREAKAGE DOESN'T OCCUR.***
  - ***TEST NEW CAPABILITIES/FEATURES ADDED TO THE BASELINE IN A CONTROLLED ENVIRONMENT.***
  - ***DELIVER THE BASELINE TO OTHER TEAMS ON THE PROGRAM TO PERFORM INTERNAL TESTING OF REQUIREMENTS AND PERFORMANCE.***
  - ***ENABLE FEATURES TO INCREMENTALLY BE TESTED BETWEEN MULTI-SUBSYSTEMS ON A PROGRAM.***

ITERATIVE (INCREMENTAL) DEVELOP

ITERATIVE DEVELOPMENT

- REQUIREMENTS NEEDED TO SUPPORT THE ITERATION ARE “FROZEN” PRIOR TO THE ITERATION STARTS
- PROJECTS TYPICALLY HAVE AT LEAST THREE INTERNAL ITERATIONS PRIOR TO THE FINAL ITERATION THAT IS RELEASED TO THE CUSTOMER.
- ITERATIONS CAN LAST FROM ONE WEEK IN DURATION TO SIX MONTHS IN DURATION ON PROJECTS THAT SPAN MULTIPLE YEARS. NOTE: THE AUTHOR STATES THAT THE RECOMMEND LENGTH OF AN ITERATION IS BETWEEN 1-6 WEEKS IN MODERN ITERATIVE METHODS.
- FEATURES/CAPABILITIES THAT DO NOT HAVE A DEPENDENCY ON A PRIOR ITERATION CAN EXECUTE IN PARALLEL.



Amount of requirement specification, design, code & test differ based on the iteration.

The author compares an iteration to a self-contained mini-project containing production-quality capabilities.

ITERATIVE PLANNING

RISK-DRIVEN ITERATIVE DEVELOPMENT

A RISK DRIVEN APPROACH TO ITERATIVE DEVELOPMENT IS BASED ON:

IDENTIFYING THE MOST CHALLENGING OR RISKY REQUIREMENTS TO IMPLEMENT IN EARLY ITERATIONS.

THESE REQUIREMENTS TYPICALLY INVOLVE INCORPORATING NEW TECHNOLOGY, PERFORMANCE REQUIREMENTS, OR OTHER RISKS IDENTIFIED DURING THE INITIAL PLANNING STAGE OF THE PROGRAM.

EXAMPLE: AN EXAMPLE PROVIDED BY THE AUTHOR INVOLVED 2 REQUIREMENTS FOR A SYSTEM:

1. THE WEBPAGES TO BE GREEN AND
2. THE SYSTEM SHALL BE ABLE TO HANDLE 5,000 SIMULTANEOUS TRANSACTIONS (WHICH SHOULD BE IMPLEMENTED 1ST)

CLIENT-DRIVEN ITERATIVE DEVELOPMENT

A CLIENT DRIVEN APPROACH TO ITERATIVE DEVELOPMENT IS BASED ON THE CUSTOMER DEFINING THE FEATURES/CAPABILITIES CONTAINED IN THE NEXT ITERATION.

ADVANTAGE OF THIS APPROACH INCLUDE:

- CUSTOMER PRIORITIZES THE FEATURES/CAPABILITIES THAT ARE THE MOST IMPORTANT TO THEM FOR EARLY ITERATIONS.
- THE APPROACH CAN BE ADAPTIVE FOR EACH ITERATION BASED ON INSIGHT THE CUSTOMER GAINS DURING THE PREVIOUS ITERATIVE DEVELOPMENT CYCLE.

TIMEBOXED ITERATIVE DEVELOPMENT

TIMEBOXING

- IS AN APPROACH IN WHICH THE ITERATION RELEASE DATE IS FIXED.
- THIS APPROACH CAN APPLY TO ONE OR ALL ITERATIONS WITHIN A PROGRAM, HOWEVER THE TIMEBOX LENGTH FOR EACH ITERATION DOES NOT NEED TO BE EQUAL.
- IF FEATURES/CAPABILITIES ASSOCIATED WITH THE ITERATION CAN NOT BE MET WITHIN THE SCHEDULED COMPLETION DATE OF THE ITERATION (TIMEBOX), THEN THE FUNCTIONALITY WITHIN THE ITERATION IS REDUCED (SCOPE MOVED TO PROGRAM BACKLOG).
- MOST ITERATIVE INCREMENTAL DEVELOPMENT (IID) METHODS RECOMMEND TIMEBOXING OF 1-6 WEEKS IN DURATION.
- THREE-MONTH TO SIX-MONTH TIMEBOXING HAS BEEN SUCCESSFULLY EXECUTED ON LARGE PROGRAMS CONTAINING HUNDRED OF SOFTWARE DEVELOPERS!

TIMEBOXED ITERATIVE DEVELOPMENT

THE PROBABILITY THAT AN ITERATIVE DEVELOPMENT APPROACH WILL BE SUCCESSFUL IS DETERMINED BY FOUR VARIABLES THAT IMPACT A PROGRAM: TIME, SCOPE, RESOURCES (STAFF AND LAB EQUIPMENT), QUALITY.

- **TIMEBOXING REMOVES THE VARIABLE OF TIME.**
- **PROCESS REMOVES THE VARIABLE OF QUALITY.**
- **THE IPTL/SPM NEEDS TO PREVENT EXTERNAL STAKEHOLDERS FROM CHANGING EITHER THE SCOPE OR THE RESOURCES ALLOCATED TO THE ITERATION ONCE IT BEGINS.**
- **THE TEAM CAN DE-SCOPE A TASK IF IT CAN NOT FIT WITHIN AN ITERATION’S TIMEBOX ONLY WITH THE APPROVAL OF THE IPTL/SPM.**

  ALTHOUGH TIMEBOXING SHOULD NOT BE USED TO HAVE SOFTWARE DEVELOPERS WORK LONGER HOURS TO HIT A PROJECT DEADLINE, IT MAY BE NECESSARY IN SOME CASES BASED ON THE IMPACT TO THE OVERALL PROGRAM.

  ALTHOUGH IT IS OK TO PUSH FEATURES/CAPABILITIES TO THE NEXT ITERATION OR TO THE PRODUCT BACKLOG, IT IS NECESSARY TO UNDERSTAND

  - **TECHNICAL DEPENDENCIES (IF THIS FEATURE IS MOVED TO A LATER ITERATION, WHAT SCHEDULE/COST IMPACT WILL THIS HAVE ON OTHER FEATURES/CAPABILITIES BEING DEVELOPED IN FUTURE INCREMENTS),**
  - **SCHEDULE DEPENDENCIES (BY MOVING THIS FEATURE TO EITHER A LATER SPRINT OR TO THE PRODUCT BACKLOG, WILL THERE BE A RESOURCE CONFLICT ON A LATER DEVELOPMENT ACTIVITY), AND**
  - **COST DEPENDENCIES (WILL THE TEAM COMPLETE THE CURRENT ITERATION WITHIN COST, OR ARE THEY PUSHING A COST-OVER RUN TO THE FUTURE).** ***FIRM-FIXED PRICE (FFP) PROGRAM VS. COST PLUS PROGRAM***

EVOLUTIONARY AND ADAPTIVE DEVELOPMENT

EVOLUTIONARY ITERATIVE DEVELOPMENT

IMPLIES THAT PROGRAM REQUIREMENTS, ESTIMATES, AND SOLUTIONS EVOLVE OR ARE REFINED OVER TIME VS. HAVING ALL REQUIREMENTS DEVELOPED UP FRONT AND FROZEN THROUGH THE COURSE OF THE ITERATIVE DEVELOPMENT LIFECYCLE.

FOCUS IS ON HIGH RISK, PERFORMANCE AND USABILITY REQUIREMENTS UP FRONT.

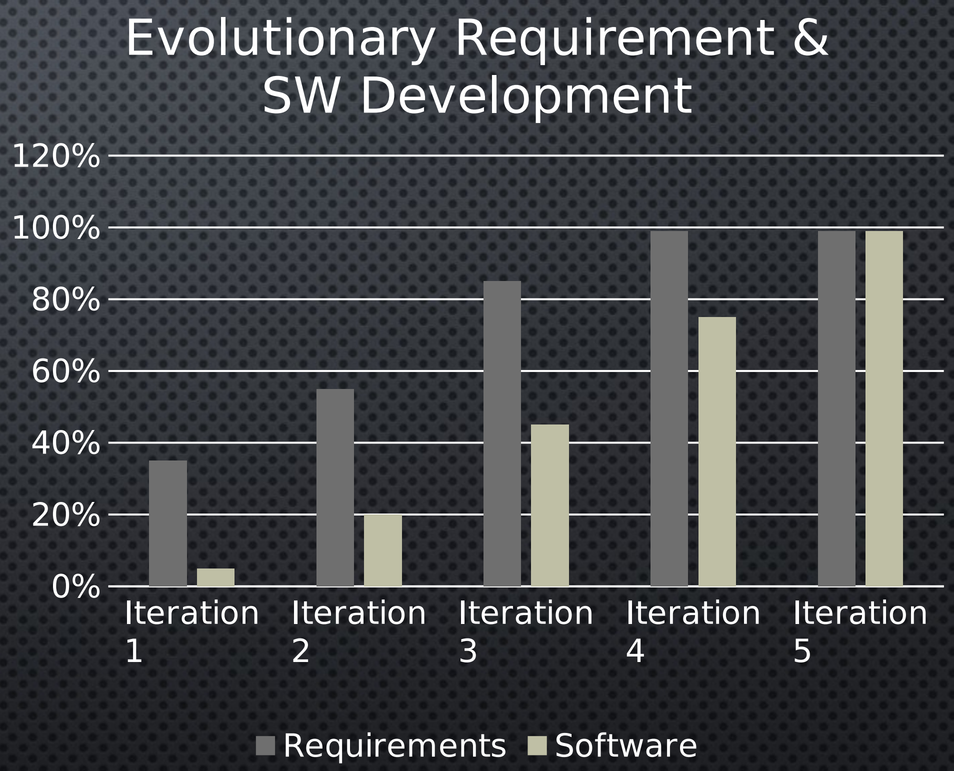
RECOMMENDATION IS TO HAVE WORKSHOPS INCLUDING SYSTEMS AND SOFTWARE ENGINEERS TO FLUSH OUT THE DETAILS.

ADAPTIVE DEVELOPMENT

IMPLIES THAT DEVELOPMENT IS ADAPTED FOR EACH ITERATION BASED ON FEEDBACK OR INSIGHT GAINED DURING THE PREVIOUS ITERATIVE DEVELOPMENT CYCLE.

DOESN'T IMPLY THAT THERE ARE NO COST AND NO SCHEDULE BOUNDARIES, JUST THAT IT IS HARDER TO PREDICT UP FRONT AND CAN BE BETTER REVISED IN FURTHER ITERATIONS.

ROLLING WAVE PLANNING IS A WAY TO ACCOMPLISH THIS BY CREATING TASKS IN THE IMS THAT ARE DETAILED PLANNED AS WORK PACKAGES DURING THE NEXT 6 MONTHS OF THE PROGRAM AND THE REMAINING TASKS ARE PLANNED AT A HIGHER LEVEL IN A PLANNING PACKAGE.



INCREMENTAL AND EVOLUTIONARY DELIVERY

INCREMENTAL DELIVERY

IS THE PROCESS OF DELIVERING A SYSTEM TO THE CUSTOMER IN A SERIES OF EXPANDED CAPABILITIES/FEATURES. TIME BETWEEN AN INCREMENTAL DELIVERY CAN RANGE BETWEEN 3-12 MONTHS.

INCREMENTAL PRODUCT DELIVERY IS NOT THE SAME AS ITERATIVE DEVELOPMENT. EACH INCREMENTAL DELIVERY CAN BE COMPOSED OF MULTIPLE ITERATIVE DEVELOPMENT CYCLES.

EVOLUTIONARY DELIVERY

IS SIMILAR TO INCREMENTAL DELIVERY EXCEPT IT CAPTURES CUSTOMER FEEDBACK AND PROVIDES THAT AS GUIDANCE INTO THE NEXT DELIVERY.

CASE STUDY (THE MOST COMMON MISTAKES?)

COMPANY X-Y-Z ACKNOWLEDGES THAT THEY HAVE CHOSEN TO USE AN ITERATIVE DEVELOPMENT METHODOLOGY SINCE THE WATERFALL LIFECYCLE MODEL IS NOT VERY SUCCESSFUL. HOWEVER, THEY STATE THAT THEY WILL NOT BEGIN SOFTWARE DEVELOPMENT UNTIL THEY COMPLETE THE USE CASE ANALYSIS, INITIAL IMP AND IMS, AND THE SYSTEM LEVEL REQUIREMENT SPECIFICATIONS.

ALTHOUGH THE AUTHOR STATES THAT THIS IS ONE OF THE MOST COMMON MISTAKES THAT NEW ITERATIVE AND AGILE METHOD ADOPTERS MAKE. I BELIEVE HIS STATEMENT IS TOO GENERIC AND SHOULD SPECIFICALLY FOCUS ON THE FACT THAT NOT ALL REQUIREMENT SPECIFICATIONS NEED TO BE COMPLETED PRIOR TO BEGINNING THE SOFTWARE DEVELOPMENT EFFORT.

AS DISCUSSED DURING LECTURE 2 – THERE ARE MANY ORGANIZATIONAL AND PROGRAMMATIC PROCESSES THAT DETERMINE YOUR ABILITY TO EFFECTIVELY ADAPT ITERATIVE/AGILE METHODOLOGY INTO THE SOFTWARE DEVELOPMENT MODEL USED ON A PROGRAM.

- LARGE PROGRAMS THAT SPAN MULTIPLE YEARS AND/OR INCLUDE MULTIPLE SUBSYSTEMS NEED SYSTEM LEVEL REQUIREMENTS TO BE WRITTEN TO A LEVEL THAT CAN BE ALLOCATED TO EACH SUB-SYSTEM. ESPECIALLY HIGH RISK, PERFORMANCE, USABILITY AND INTERFACE REQUIREMENTS PRIOR TO STARTING SOFTWARE DEVELOPMENT IN THOSE AREAS.
- A FIRM-FIXED PRICE (FFP) CONTRACT IS A BINDING AGREEMENT BETWEEN YOUR COMPANY AND YOUR CUSTOMER THAT YOU WILL DELIVER THE REQUIREMENTS SPECIFIED IN THE CONTRACT WITHIN A SPECIFIED SCHEDULE AND FOR A SPECIFIED COST. THE CONTRACT MAY EVEN SPECIFY THAT YOU WILL CONDUCT A PRELIMINARY DESIGN REVIEW (PDR) DURING MONTH 3 OF THE CONTRACT AND A DETAILED DESIGN REVIEW DURING MONTH 9 OF THE CONTRACT WHICH WILL ULTIMATELY DRIVE WHAT THE FOCUS IS DURING THE EARLY ITERATIONS OF THE DEVELOPMENT LIFECYCLE.
- IN ADDITION, CUSTOMER INVOLVEMENT DURING ITERATIVE DEVELOPMENT MAY NOT BE DESIRABLE IF THE CONTRACT IS FIRM-FIXED PRICE AND THERE IS NO CONTRACTUAL MECHANISM TO ELIMINATE LESS IMPORTANT REQUIREMENTS SPECIFIED IN THE CONTRACT DURING THE ITERATIVE DEVELOPMENT LIFECYCLE.
- REMEMBER THE ONLY WAY TO MANAGE THE PROGRAM SCHEDULE IS TO HAVE A DETAILED IMS THAT CRITICAL PATH ANALYSIS CAN BE USED TO IDENTIFY WHAT IS DRIVING THE PROGRAM SO YOU CAN MANAGE YOUR SCHEDULE BUFFER. ALSO DETAILED SOFTWARE PLANNING ENABLES YOU TO CREATE AN INITIAL COST BASIS AT THE BUILD OR ITERATION LEVEL THAT CAN BE REFINED AS YOU GO.

SPECIFIC ITERATIVE & EVOLUTIONARY METHODS

UNIFIED PROCESS (UP)

UP OR RATIONAL UNIFIED PROCESS IS THE MOST WIDELY USED ITERATIVE APPROACH ACROSS THOUSANDS OF ORGANIZATIONS WORLDWIDE.

IT WAS DEVELOPED IN THE MID-1990'S WITH INPUTS FROM MANY EXPERIENCED SYSTEM ARCHITECTS AND DESIGNERS.

ORGANIZATIONS THAT USE THE UP METHOD TYPICALLY FOCUS ON THE CORE ARCHITECTURE OF A SYSTEM AND HIGH-RISK AREAS OF DEVELOPMENT IN EARLY ITERATIONS TO MINIMIZE RISK TO THE PROGRAM.

EVOLUTIONARY (EVO)

EVOLUTIONARY PROCESS WAS DEVELOPED IN THE 1960'S WITH THE FOCUS BEING SHORT ITERATIONS OF 1-2 WEEKS IN DURATION

EVO USES ADAPTIVE PLANNING AND FOCUSES ON THE HIGHEST VALUE-TO-COST RATION ITEMS FIRST.

EVO ALSO PROMOTES THE USE OF UNAMBIGUOUS AND QUALITY REQUIREMENTS THAT CAN BE QUANTIFIED OR MEASURED (IN OTHER WORDS REQUIREMENTS MUST BE TESTABLE).

REFERENCES

AGILE & ITERATIVE DEVELOPMENT, A MANAGER’S GUIDE, CRAIG LARMAN, EIGHTH EDITION, ADDISON WESLEY, NEW YORK, NY, COPYRIGHT 2004 BY PEARSON EDUCATION , INC.

PROCESS

AGILE & ITERATIVE DEVELOPMENT (CHAPTER 3)

A MANAGER'S GUIDE BY: CRAIG LARMAN

WEEK 4: AGILE METHODS

BY: JOSEPH MARTINAZZI

AGILE DEVELOPMENT

AGILE DEVELOPMENT HAS A MOTTO “EMBRACE CHANGE”

- **AGILE DEVELOPMENT METHODS ARE A SUBSET OF ITERATIVE AND EVOLUTIONARY DEVELOPMENT METHODS.**
- **ALTHOUGH AGILE METHODS VARY IN STYLE, THEY ALL EMPLOY SHORT TIMEBOXED ITERATIONS WITH EVOLUTIONARY PLANNING TECHNIQUES.**
- **AGILE DEVELOPMENT EMPHASIZES SIMPLICITY, DIRECT COMMUNICATIONS, SELF-DIRECTED TEAMS, AND WORKING CODE.**

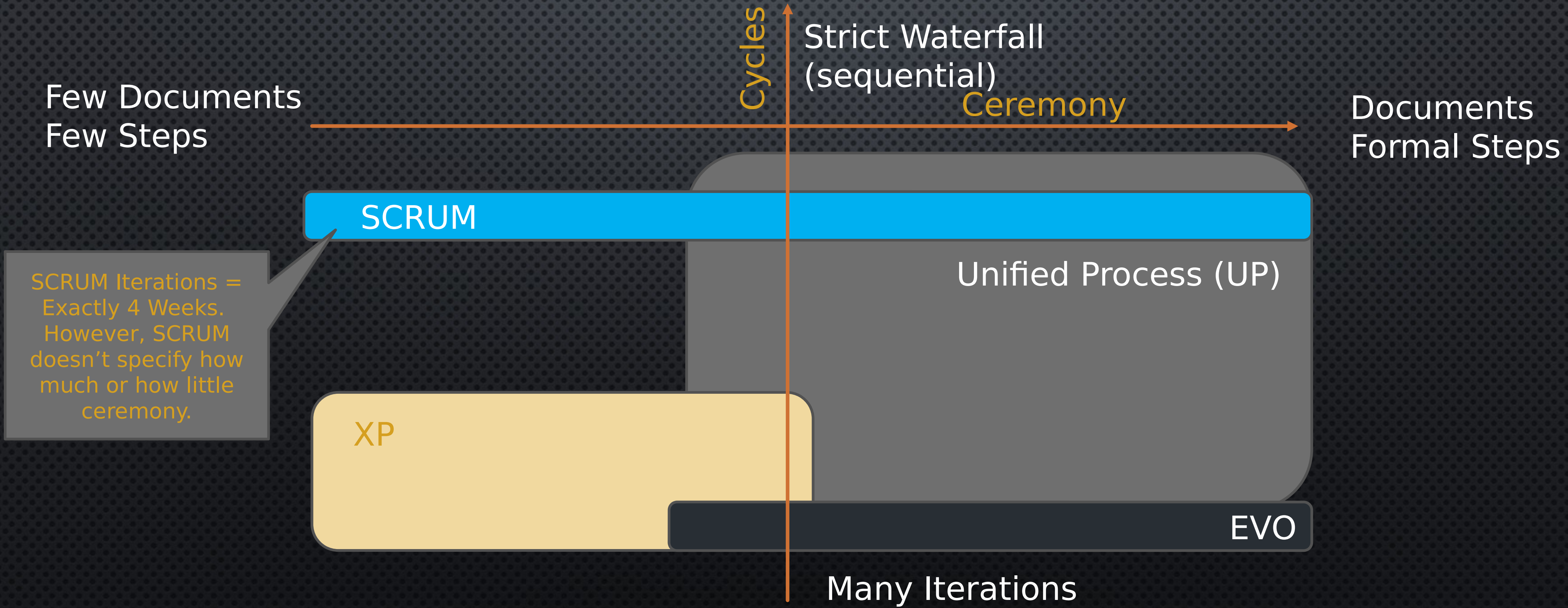
  A SCRUM EXAMPLE: INVOLVES SELF-DIRECTED TEAMS WORKING IN A COMMON PROJECT ROOM. EACH TEAM COORDINATES ACTIVITIES AND PROGRESS VIA DAILY STANDUP MEETINGS IN WHICH EACH TEAM MEMBER IS HELD ACCOUNTABLE BY RESPONDING TO A SET LIST OF QUESTIONS.

  AN XP EXAMPLE: INVOLVES SELF-DIRECTED TEAMS CONTAINING A DEDICATED SE TO EXPLAIN/DECOMPOSE REQUIREMENTS TO GROUPS OF PAIRED-PROGRAMMERS WORKING IN A COMMON ROOM.

CLASSIFICATION OF METHODS

AGILE METHODS ARE CLASSIFIED BASED ON “CEREMONY” AND “CYCLES”.

- CEREMONY – IS THE AMOUNT OF DOCUMENTATION, FORMAL STEPS, REVIEWS, ETC.
- CYCLES – IS THE NUMBER AND LENGTH OF ITERATIONS.



Cycles

Strict Waterfall
(sequential)

Ceremony

Few Documents
Few Steps

Documents
Formal Steps

SCRUM

Unified Process (UP)

XP

EVO

Many Iterations

SCRUM Iterations = Exactly 4 Weeks. However, SCRUM doesn’t specify how much or how little ceremony.

THE AGILE MANIFESTO

IN 2001 A GROUP OF INDIVIDUALS DEFINED THE AGILE MANIFESTO AND A SET OF PRINCIPLES THAT AGILE TECHNIQUES SHOULD TAKE INTO CONSIDERATION.

WWW.AGILEALLIANCE.COM

THE AGILE PRINCIPLES

IN 2001 A GROUP OF INDIVIDUALS DEFINED THE AGILE MANIFESTO AND A SET OF PRINCIPLES THAT AGILE TECHNIQUES SHOULD TAKE INTO CONSIDERATION.

WWW.AGILEALLIANCE.COM

AGILE PROJECT MANAGEMENT

THE JOB OF THE AGILE PROJECT MANAGER IS TO PROMOTE THE VISION BY HAVING OPEN COMMUNICATION AND AVOIDING (MINIMIZING) COMMAND AND CONTROL.

JIM HIGHSMITH SUMMARIZES 9 PRINCIPLES FOR AGILE PROJECT MANAGEMENT.

HIGHSMITH, J. 2002. AGILE SOFTWARE DEVELOPMENT ECOSYSTEMS. ADDISON-WESLEY

RECOMMENDATIONS PROJECT MANAGEMENT

AUGUSTINE AND WOODCOCK, RECOMMEND THE FOLLOWING 6 PRACTICES FOR AGILE PROJECTS.

AUGUSTINE, S. AND WOODCOCK. S. 2002. “AGILE PROJECT MANAGEMENT: EMERGENT ORDER THROUGH VISIONARY LEADERSHIP.” CC PACE SYSTEMS. JULY 2002.

AGILE PROJECT MANAGEMENT

THE AUTHOR STATES THAT THE ENTIRE TEAM NEEDS TO PARTICIPATE IN CONTROL AND PLANNING ON AGILE PROJECTS. THE MANAGER DOESN'T CREATE THE WBS, DEFINE THE SCHEDULE, AND ESTABLISH BUDGETS, BUT THE TEAM DOES!

- ALTHOUGH THIS MAY WORK FOR SMALL TEAMS IN WHICH COST AND SCHEDULE DO NOT MATTER, IT WILL NOT WORK FOR LARGE SCALE SOFTWARE DEVELOPMENT EFFORTS OR FIRM FIXED PRICE CONTRACTS.
- REMEMBER THE IMP, A HIGH-LEVEL IMS AND THE WBS STRUCTURE ARE TYPICALLY CONTAINED IN A COMPANY'S PROPOSAL IN RESPONSE TO A CUSTOMER'S RFP.

AGILE PROJECT MANAGEMENT

SO, DOES THAT MEAN AGILE DEVELOPMENT SHOULD NOT BE USED ON A LARGE-SCALE SOFTWARE DEVELOPMENT EFFORT? ABSOLUTELY NOT!

THE TEAM CAN STILL PLAN HOW THEY

- *DEVELOP THE USE CASE INCLUDING DEFINING THE BACKLOG, AND*
- *VALIDATE THE BUDGET ALLOCATED TO THE USE CASE.*

THE MANAGER'S JOB IS TO:

- *DETERMINER THE SIZE OF THE TEAM NEEDED TO COMPLETE THE WORK WITHIN THE TIMEBOX,*
- *COMMUNICATE WITH THE TEAM IN ORDER TO UNDERSTAND THEIR ISSUES,*
- *REMOVE ROADBLOCKS IMPEDING THE TEAM'S PROGRESS,*
- *VALUE AGILE PRINCIPLES BY FEEDING BACK RECOMMENDATIONS FROM THE TEAM TO STREAM-LINE PROCESS, REMOVE ROADBLOCKS, ETC., AND*
- *BECOME CAPTAIN AMERICA AND SHIELD THE TEAM FROM OUTSIDE INFLUENCES.*

AGILE PROJECT DEVELOPMENT

EMBRACE CHANGE

THE AGILE PRIME DIRECTIVE IS TO EMBRACE CHANGE BY BEING ADAPTIVE.

EMBRACE COMMUNICATIONS AND FEEDBACK

AGILE DEVELOPMENT STRIVES TO INCREASE DAILY FACE-TO-FACE COMMUNICATION THROUGH DAILY MEETINGS (SCRUM) OR BY HAVING A CUSTOMER PRESENT IN THE COMMON PROJECT ROOM (XP).

AGILE DEVELOPMENT IS CONSTANTLY ADAPTING BY PROVIDING DEMOS TO GET CUSTOMER FEEDBACK ON PRODUCT DEVELOPMENT AND TEAM FEEDBACK ON WHAT WORKS AND WHAT DOESN’T WORK (PROCESS).

PROGRAMMING AS IF PEOPLE MATTERED

A HAPPY TEAM IS MORE PRODUCTIVE WHICH RESULTS IN SUSTAINABLE DEVELOPMENT

MAINTAIN A WORK/LIFE BALANCE AND MINIMIZE OVERTIME

KNOWLEDGE AND WORK HABITS PLAY A SIGNIFICANT ROLE IN AN INDIVIDUAL'S PRODUCTIVITY

MENTOR NEW TEAM MEMBERS

SIMPLICITY IS A KEY TO SUCCESS

DO THINGS THE SIMPLEST WAY POSSIBLE, AVOID HIGH TECH SOLUTIONS IF POSSIBLE.

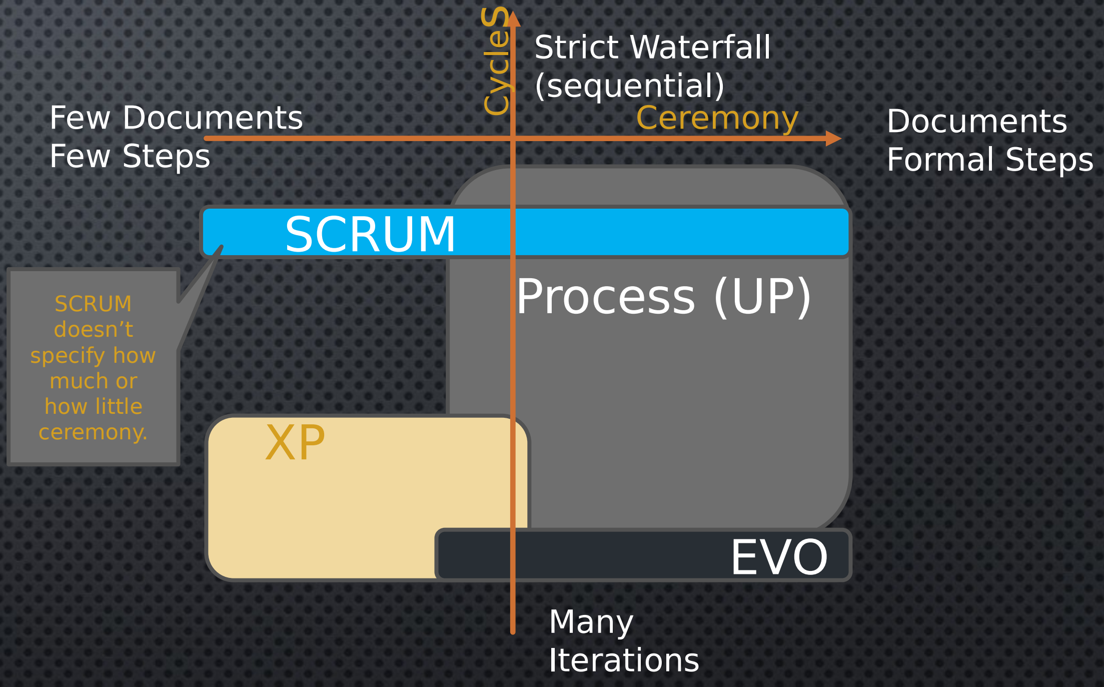
AGILE PROJECT DEVELOPMENT

EMPIRICAL VS. DEFINED (PRESCRIPTIVE) PROCESS

THE AUTHOR STATES THAT DEFINED PROCESSES ARE SUITABLE FOR PREDICTABLE MANUFACTURING DOMAINS AND THAT EMPIRICAL PROCESS IS MORE FOR AGILE DEVELOPMENT. HOWEVER, HE ALSO STATES THAT SCRUM DOESN'T SPECIFY THE AMOUNT OF CEREMONY.

A DEFINED PROCESS – HAS MANY PREDEFINED AND SEQUENTIAL ACTIVITIES.

AN EMPIRICAL PROCESS – ARE BASED ON FREQUENCY MEASUREMENT AND DYNAMIC RESPONSES TO VARIABLE EVENTS *(E.G. , AGILE PRINCIPLES 12 & 13).*



SPECIFIC AGILE METHODS

SCRUM

SCRUM EMPHASIZES SELF-ORGANIZED TEAMS, WITH DAILY STANDUP MEETINGS (COMMUNICATION/FEEDBACK), AND DAILY TEAM MEASUREMENT (PEER PRESSURE TO DRIVE PERFORMANCE).

SCRUM ITERATIONS ARE 4 WEEKS IN DURATION, WITH A DEMO TO EXTERNAL STAKEHOLDERS AT THE END OF THE ITERATION.

XP

XP EMPHASIZES COLLABORATION (VIA PEER PROGRAMMING, TEAM WORKING IN A COMMON PROJECT ROOM), CONSTANT REFACTORING OF THE CODE, AND TEST-DRIVEN DEVELOPMENT *(PRACTICE OF DEVELOPING TEST CASES PRIOR TO DEVELOPING THE CODE)*.

IT IS FOUNDED ON 4 VALUES: COMMUNICATION, SIMPLICITY, FEEDBACK, AND COURAGE.

CRYSTAL FAMILY OF AGILE METHODS

DEVELOPED BY ALISTAIR COCKBURN

- DEFINES PROJECT COMPLEXITY BASED ON THE CRITICALITY OF THE END-PRODUCT AND SIZE OF STAFF REQUIRED TO COMPLETE THE PROJECT.
  - PROCESS CEREMONY (DEFINED STEPS, DOCUMENTATION, REVIEWS, ETC.) INCREASE BASED ON THESE FACTORS.
  - DEVELOPED A CLASSIFICATION MODEL TO ASSIST IN SCALING PROGRAMS.
    - AN E6 EQUATES TO A PROJECT REQUIRING A STAFF OF 1-6 INDIVIDUALS AND IN WHICH A FAILURE WOULD RESULT IN A LOSS OF ESSENTIAL MONEY.
    - AN L100 EQUATES TO A PROJECT REQUIRING A STAFF OF 41-100 INDIVIDUALS AND IN WHICH A FAILURE WOULD RESULT IN A LOSS OF LIFE!

This classification model is used to identify methodologies best suited for UP, SCRUM, XP, and/or Evo process models.

Life-Critical

Company Fails

Lost Profits

Annoyance

REFERENCES

AGILE & ITERATIVE DEVELOPMENT, A MANAGER’S GUIDE, CRAIG LARMAN, EIGHTH EDITION, ADDISON WESLEY, NEW YORK, NY, COPYRIGHT 2004 BY PEARSON EDUCATION , INC.

CS466 – SOFTWARE PROCESS

AGILE & ITERATIVE DEVELOPMENT (CHAPTERS 5&4)

A MANAGER'S GUIDE BY: CRAIG LARMAN

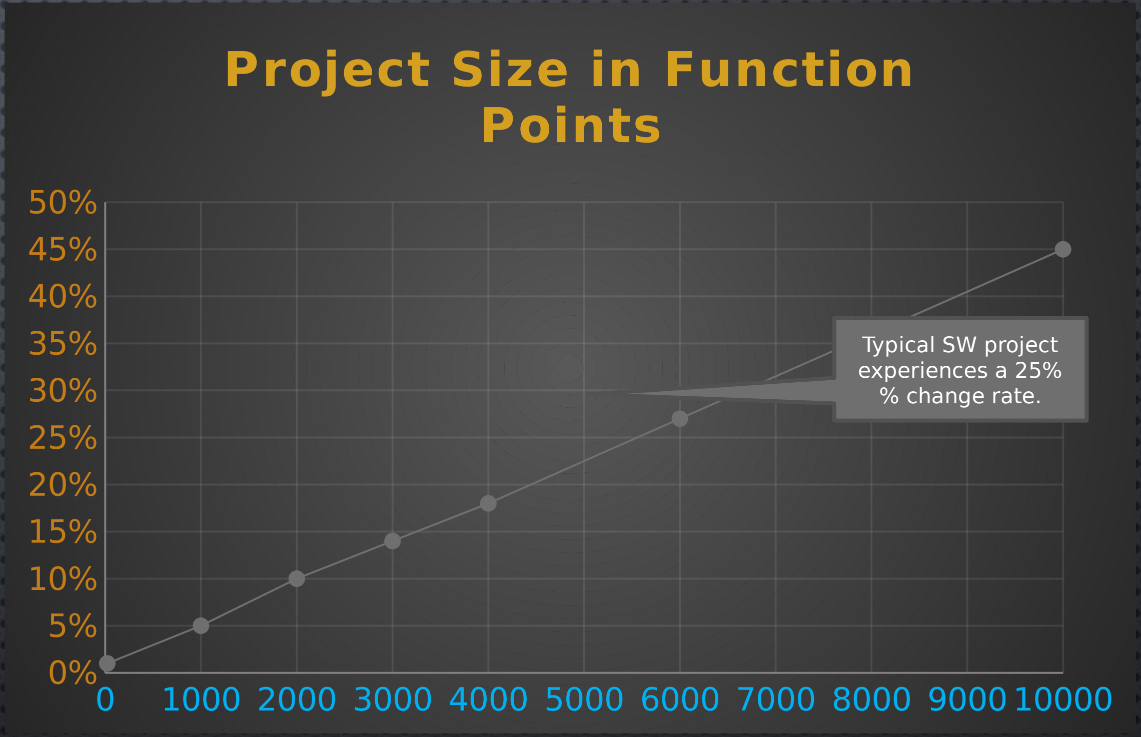
WEEK 5: MOTIVATION & AN AGILE CASE STUDY

BY: JOSEPH MARTINAZZI

THE FACTS OF CHANGE ON A SW PROJECT

THE FOLLOWING GRAPH IS BASED ON RESULTS FROM MULTIPLE LARGE-SCALE SOFTWARE DEVELOPMENT PROJECTS. [JONES97]

- IT ILLUSTRATES THAT AS THE COMPLEXITY OF THE PROJECT INCREASES (FUNCTION POINTS) THE AMOUNT OF REQUIREMENT CHANGE (OR CREEP) ALSO INCREASES.
- MEDIUM SIZE PROJECTS HAVE A CHANGE RATE OF 25%
- LARGE SIZE PROJECTS HAVE A CHANGE RATE OF 35%



Typical SW project experiences a 25% % change rate.

THIS ENFORCES THE CONCEPT THAT AN ITERATIVE LIFECYCLE MODEL HAS A BETTER CHANCE OF SUCCESS THAN A SEQUENTIAL LIFECYCLE MODEL SINCE IT CAN ADAPT BETTER TO CHANGING REQUIREMENTS, FOCUSES ON ARCHITECTURE AND HIGH RISK REQUIREMENTS EARLY, HAS A BETTER PRODUCTIVITY RATE, AND A PRODUCES A HIGHER QUALITY PRODUCT (ONE WITH FEWER DEFECTS).

KEY MOTIVATIONS FOR ITERATIVE DEVELOPMENT

Iterative life-cycle models compared to sequential life-cycle models	
The iterative life-cycle model is lower risk	*compared to the waterfall life-cycle model.*
The iterative life-cycle model is designed for early risk mitigation and discovery	*compared to the waterfall life-cycle model.*
The iterative life-cycle model supports the high-change nature of software development	*compared to the waterfall life-cycle model.*
The iterative life-cycle model builds team and customer confidence as production code is incrementally released	*compared to the waterfall life-cycle model.*
The iterative life-cycle model provides opportunity to demo the system to other potential customers	*compared to the waterfall life-cycle model.*
The iterative life-cycle model provides more relevant project tracking	*compared to the waterfall life-cycle model.*
The iterative life-cycle model provides a higher quality product with less defects	*compared to the waterfall life-cycle model.*
The iterative life-cycle model provides a higher probability that the final product will be want the customer wants	*compared to the waterfall life-cycle model.*
The iterative life-cycle model better supports the concept of continual process improvement	*compared to the waterfall life-cycle model.*
The iterative life-cycle model requires more customer engagement, resulting a better probability of success	*compared to the waterfall life-cycle model.*

KEY MOTIVATION FOR TIMEBOXING

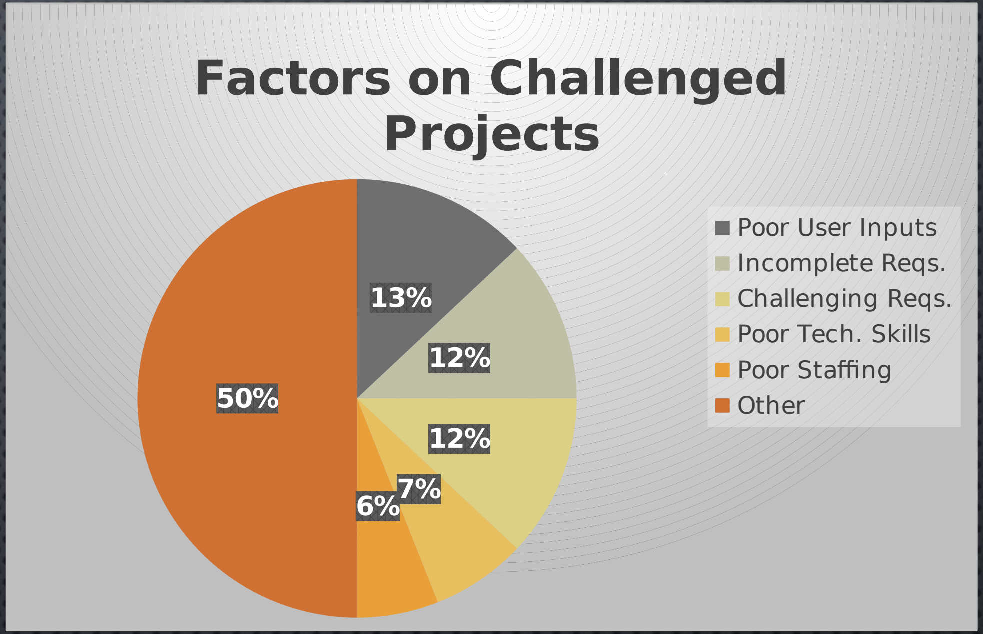
- THE PRACTICE OF TIMEBOXING INCREASES PRODUCTIVITY AS A RESULT OF FOCUSING THE TEAM ON THE END DATE OF THE TIMEBOX. THE AUTHOR STATES THAT TIMEBOXING MAY BE VIEWED AS AN ANTIDOTE TO PARKINSON'S LAW: "WORK EXPANDS SO AS TO FILL THE TIME AVAILABLE FOR ITS COMPLETION." [PARKINSON58]
- ANOTHER BENEFIT OF TIMEBOXING ITERATIONS AS WELL AS THE ENTIRE PROJECT IS BECAUSE PEOPLE REMEMBER SLIPPED DATES, *BUT NOT SLIPPED FEATURES*. EVERYONE WILL VIEW A PROJECT THAT SLIPS 3 MONTHS HAVING 100% OF THE FUNCTIONAL AS A "FAILURE", HOWEVER THE PERCEPTION OF A PROJECT THAT DELIVERS 75% OF THE FUNCTIONAL ON TIME MAY BE CONSIDERED A SUCCESS *IN SOME CASES (E.G., WITH CUSTOMER BUY-IN)*
- ANOTHER BENEFIT OF TIMEBOXING IS IT FOCUSES THE TEAM ON TACKLING SMALL LEVELS OF COMPLEXITY WITHIN A SHORT PERIOD OF TIME.
- ANOTHER BENEFIT OF TIMEBOXING IS IT ENABLES EARLY FORCING OF DIFFICULT DECISIONS AND TRADE-OFFS.

MEETING THE REQUIREMENTS CHALLENGE ITERATIVELY

IN A STUDY OF OVER 8,000 SOFTWARE PROJECTS, 37% OF THE FACTORS ON CHALLENGED PROGRAMS RELATED TO REQUIREMENTS AS SHOWN IN THE GRAPH ON THE RIGHT (POOR USER INPUTS, INCOMPLETE REQUIREMENTS, CHALLENGING REQUIREMENTS). [STANDISH94]

IN A STUDY OF FAILURE FACTORS OF OVER 1,000 SOFTWARE PROJECTS, 82% OF THE PROJECTS SITED REQUIREMENTS AS THE NUMBER 1 PROBLEM. [THOMAS01]

VARIOUS OTHER STUDIES SUPPORT THE FACT THAT REQUIREMENT CREEP IS A LARGE CONTRIBUTOR TO PROJECT FAILURE.



PROPONENTS OF THE WATERFALL METHOD – TYPICALLY POINT TO THIS REASON AS WHY IT IS ESSENTIAL TO FREEZE REQUIREMENT DEVELOPMENT UP-FRONT.

HOWEVER, THIS IS EXACTLY WHY ITERATIVE INCREMENTAL DEVELOPMENT OF REQUIREMENTS WORK – IT FORCES THE CHANGE TO OCCUR EARLY IN THE PROJECT, THUS MINIMIZING THEIR IMPACT!

PROBLEMS WITH THE WATERFALL METHODOLOGY

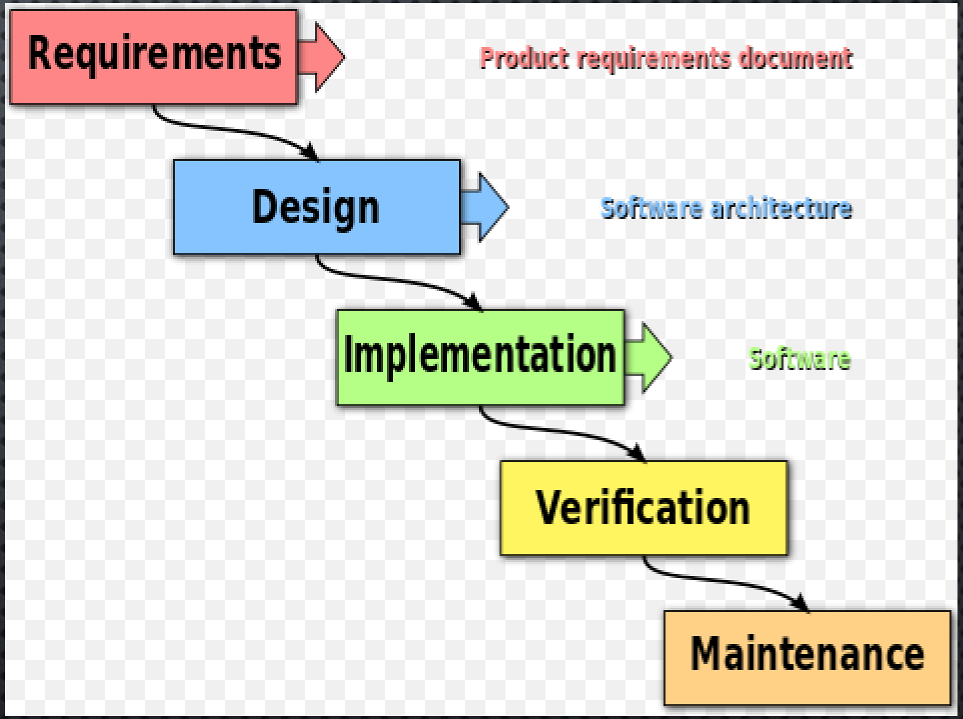
THE COMMON USAGE OF THE WATERFALL LIFECYCLE MODEL WAS SEQUENTIALLY FOLLOWING THE STEPS OF REQUIREMENTS, DESIGN, IMPLEMENTATION, VERIFICATION, AND MAINTENANCE.

1. DEFINE ALL REQUIREMENTS IN DETAIL UP-FRONT
2. DEFINE THE SYSTEM IN “TEXT” AND “DIAGRAMS”
3. IMPLEMENT THE SYSTEM “CODE, UNIT TEST, INTEGRATE”
4. INTEGRATE AND TEST THE SYSTEM COMPONENTS.

THIS MODEL DOES NOT WORK WELL WITH ADAPTING REQUIREMENTS.

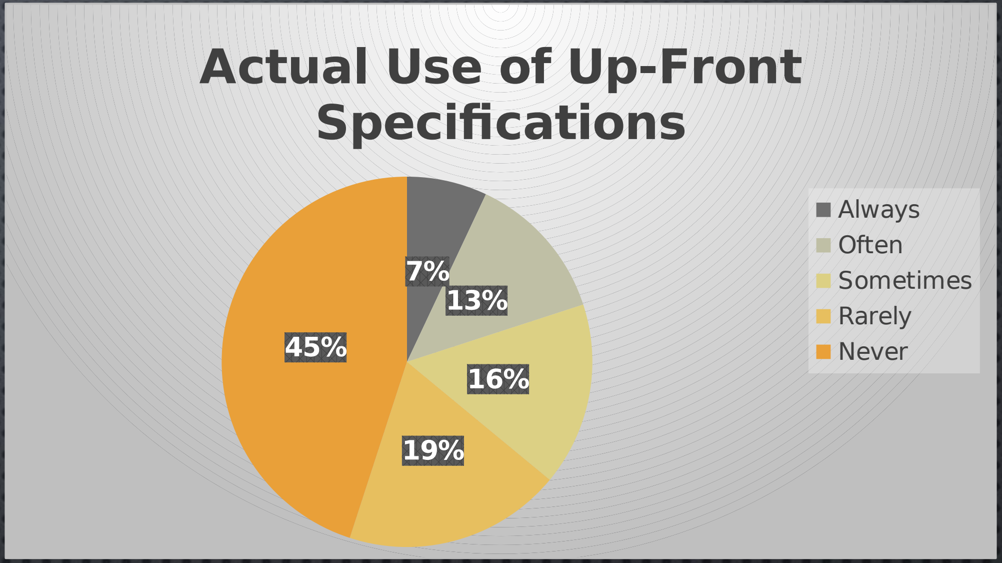
ALTHOUGH THIS WAS THE PREFERRED METHOD OF MANAGING A SOFTWARE PROJECT IN THE 1970S, TODAY'S RESEARCH CLEARLY SHOWS THAT THIS METHODOLOGY IS ASSOCIATED WITH HIGHER RISK, HIGHER FAILURE RATES, AND LOWER PRODUCTIVITY.

IN ADDITION, THE WATERFALL APPROACH RESULTS IN OVERWHELMING DEGREES OF COMPLEXITY SINCE IT DOESN’T BREAK THE DEVELOPMENT INTO MORE MANAGEABLE LEVELS OF COMPLEXITY (E.G., A SUBSET OF CAPABILITIES)



PROBLEMS WITH DEVELOPING UP-FRONT REQUIREMENTS

- IN ANOTHER STUDY THE AUTHOR STATES THAT UP-FRONT SPECIFICATION WITH A SIGN-OFF CAN NOT BE SUCCESSFULLY CREATED AND THAT A STUDY SHOWED THAT 45% OF THE FEATURES CREATED FROM EARLY SPECIFICATION WERE NEVER USED, WITH AN ADDITIONAL 19% RARELY USED AS SHOWN IN THE GRAPH ON THE RIGHT. [JOHNSON02]
- THE AUTHOR THEN PROCEEDED TO SAY "AVOID PREDICTIVE PLANNING BECAUSE YOU CAN NOT SIMPLY PLAN THE WORK AND WORK THE PLAN" WHEN DOING ITERATIVE SOFTWARE DEVELOPMENT.
- THIS WILL ONLY WORK IF YOU PROJECT IS NOT FIRM FIXED PRICE OR IF YOUR CUSTOMER HAS BOUGHT INTO THE IDEA OF YOU DELIVERING A SYSTEM WITH ONLY 75%-95% OF THE FEATURES THEY CONTRACTED!



AN AGILE CASE STUDY

AN AGILE PROJECT EXAMPLE - THE "STORY" OVERVIEW

IN CHAPTER 4 OF THE TEXT, THE AUTHOR PROVIDES AN EXCELLENT EXAMPLE OF USING A VARIETY OF AGILE TECHNIQUES (UP, EVO, SCRUM, AND XP) TO MANAGE A PROGRAM

- COMPANY: BORDER INFORMATION GROUP (BIG)
- PROJECT: BIOMETRIC RECORDING OR TRACKING HAZARDOUS EXTERNAL RADICALS (BROTHER)
- PROJECT MANAGER: CONVINCED UPPER MANAGEMENT THAT THE BEST WAY TO IMPLEMENT THIS PROJECT WAS TO USED TIMEBOXED ITERATIVE DEVELOPMENT COMBINED WITH TIMEBOXED INCREMENTAL DELIVER.
- IMPLEMENTATION TEAM: 1 PROJECT MANAGER, 1 SYSTEM ARCHITECT, 5 SOFTWARE DEVELOPERS
- *PROJECT START DATE = 1/1/2021…. 1ST TIMEBOXED INCREMENTAL DELIVERY (ID) TO CUSTOMER = 10/1/2021. DELIVERY DATA IS FIXED; OK FOR FEATURES TO FALL OUT OF 1ST DELIVERY TO CUSTOMER.*
  (REFER TO LECTURE 3, SLIDE 9 FOR DEFINITION OF ID)
- *THE CUSTOMER WILL BE AVAILABLE PART TIME EACH DAY. IN ADDITION, THERE WILL BE A DEDICATED SUBJECT MATTER EXPERT (SME) WHO'S PREVIOUS OCCUPATION OF BEING A BOARDER GUARD WILL BE AN ASSET TO THE TEAM.*
- *THE INITIAL SOFTWARE WILL BE DEPLOYED AT 2 LOW TRAFFIC AIRPORTS FOR 2 MONTHS TO GET THE BOARDER GUARD'S AND PASSENGER FEEDBACK ON SYSTEM.*

AN AGILE CASE STUDY

AN AGILE PROJECT EXAMPLE - THE “STORY” – WEEK 1

[SCRUM-01] TEAM RELOCATES TO A FACILITY AT 1 OF THE TARGET AIRPORTS THAT HAS A LARGE ROOM THAT COULD BE USED FOR COLLABORATION AND CUBICLES THAT CAN BE USED WHEN MEMBERS OF THE TEAM NEED QUIET TIME.

[SCRUM-02] TEAM TO PROVIDE A DEMO TO BIG’S UPPER MANAGEMENT EVERY 3-4 WEEKS.

[XP-01] CUSTOMER TO BE PRESENT EVERY MORNING, BOARDER GUARD TO PARTICIPATE AS SME FOR TEAM.

AN AGILE CASE STUDY

AN AGILE PROJECT EXAMPLE - THE “STORY” – WEEK 1 (CONTINUED)

[UP-01] TEAM TO HOLD A 2-DAY PLANNING AND REQUIREMENT WORKSHOP. GOAL IS TO BRAINSTORM REQUIREMENTS WHILE INCORPORATING A 20-PAGE WISH LIST FROM THE CUSTOMER.

- PROJECT MANAGER RECOMMENDS TEAM SELECT TOP 20% OF THE REQUIREMENTS AND CUSTOMER RECOMMENDATIONS BASED ON ARCHITECTURAL SIGNIFICANCE, RISK, AND VALUE. TEAM USED A DOT SYSTEM TO PRIORITIZE.
- TEAM SPENDS NEXT 2-DAYS ANALYZING REQUIREMENTS:
  - [UP-02] TEAM DECOMPOSED FUNCTIONAL REQUIREMENTS INTO MULTIPLE USE CASES
  - [EVO-01] TEAM IDENTIFIED NON-FUNCTIONAL (CUSTOMER) REQUIREMENTS THAT NEED TO BE QUANTIFIED (E.G., FAST RESPONSE) AND MEASURABLE (EASY TO USE) AS KEY REQUIREMENTS.
- TEAM LEAD SET EXPECTATIONS - FOR FIST ITERATIVE DEVELOPMENT CYCLE THAT WOULD START ON 01-09 AND END ON 01-26 WITH A DEMO CONSISTING OF A PARTIALLY RUNNING SYSTEM CONNECTED TO A BIOMETRIC METER.
  - [XP-02] TEAM DECIDES WHAT THEY CAN ACCOMPLISH WITH THE NEXT TWO WEEKS FROM THE 20% OF THE IDENTIFIED REQUIREMENTS
  - [UP-03] TEAM DECIDES TO IMPLEMENT A THE “POSITIVE PATH” ON A FEATURE THAT WILL TOUCH ON VARIOUS ARCHITECTURAL FEATURES OF THE SYSTEM.
  - [XP-03] TEAM DETERMINES THE NUMBER OF HOURS NEEDED TO IMPLEMENT THE WORK AND COMPARES IT TO THE NUMBER OF AVAILABLE HOURS WITHIN THE TIME BOX (ASSUMING NO OVERTIME). THE TEAM REDUCES THE SCOPE WITHIN THIS ITERATION TO FIT WITHIN THE TIMEBOX.
  - [SCRUM-03] PROGRAM MANAGER ENTERS FEATURES TARGETED FOR 1ST ITERATION INTO A SCRUM SPRINT BACKLOG SHEET.

AGILE CASE STUDY

AN AGILE PROJECT EXAMPLE - THE “STORY” – WEEK 2

[SCRUM-04] TEAM HOLDS DAILY 20 MIN. STAND-UP MEETINGS: REVIEWS GOAL FOR ITERATION, REMAINING TASKS WITHIN ITERATION, HOLDS TEAM Q&A, ASKS TEAM MEMBERS TO VOLUNTEER FOR ONE OF THE REMAINING TASKS TO COMPLETE.

[UP-04] CHIEF ARCHITECT EDUCATIONS TEAM ON POTENTIAL ISSUES AND DESIGN AND EXPLAINS THEIR VISION SO THE SYSTEM CAN BE DECOMPOSED INTO COMPONENTS. TEAM REFINES IDEA AND EXPLORES AND COORDINATES THE DESIGN IDEAS ON WHITE BOARD.

[XP-04] TEAM MOVES OUT ON CODING AFTER DECIDING TO USE XP PRACTICE OF TEST-DRIVEN DEVELOPMENT. ONE OF THE DEVELOPERS IS ASSIGNED THE TASK OF DEVELOPING ACCEPTANCE TEST. AS CLASSES AND UNIT TESTS ARE CREATED, THEY ARE CHECKED INTO A BUILD MACHINE THAT RUNS THE TESTS AS PART OF CONTINUOUS INTEGRATION WHICH RESULT IN PROBLEMS BEING QUICKLY IDENTIFIED AND RESOLVED.

[XP-05] EACH MORNING A TEAM MEMBER COLLECTS METRICS ON EVERYONE’S PROGRESS AND UPDATES THE SPRINT BACKLOG SPREADSHEET. COMPLETED TASKS ARE CROSSED OUT ON THE WHITE BOARD.

AGILE CASE STUDY

AN AGILE PROJECT EXAMPLE - THE “STORY” - WEEK 3+

AS CODE FROM MULTIPLE DEVELOPERS COME TOGETHER, THE TEAM BEGINS TO DEVELOP A SYNERGY AND THE OVERALL SYSTEM STARTS TO TAKE SHAPE AS PRODUCTION CODE AND UNIT TESTS ARE CHECKED IN DAILY.

AS THE TEAM APPROACHES THE TARGET DEMO DATE, THEY DO A CHECK OF THE BACKLOG AND DETERMINE IT THAT THEY HAVE ENOUGH TIME TO COMPLETE ALL OF THE ITEMS IN TIME FOR THE DEMO.

[SCRUM-05] TEAM HOLDS A DEMO TO THE BIG EXECUTIVES. EVEN THOUGH THE SYSTEM DOESN’T DO MUCH IT WAS IMPRESSIVE THAT THERE WAS A WORKING SYSTEM WITHIN 3 WEEKS.

THE BIG EXECUTIVE REQUEST THAT THE SYSTEM MUST ALSO INTERFACE WITH A 3RD PARTY FACE RECOGNITION SYSTEM BASED ON COMPETITIVE SYSTEMS CURRENTLY UNDER DEVELOPMENT.

THE TEAM BEINGS PLANNING ITS FEATURES OF THE SECOND ITERATION THAT WILL FOCUS ON THIS HIGH PRIORITY REQUEST.

AGILE CASE STUDY

WHERE WOULD YOU CONSIDER THIS EXAMPLE TO FALL WITHIN THE COCKBURN SCALE?

This classification model is used to identify methodologies best suited for UP, SCRUM, XP, and/or Evo process models.

L20 ?

E20 ?

D20 ?

Life-Critical

Company Fails

Lost Profits

Annoyance

REFERENCES

AGILE & ITERATIVE DEVELOPMENT, A MANAGER’S GUIDE, CRAIG LARMAN, EIGHTH EDITION, ADDISON WESLEY, NEW YORK, NY, COPYRIGHT 2004 BY PEARSON EDUCATION , INC.

SPECIFIC SOURCES THE AUTHOR QUOTED:

[JOHNSON02] - JOHNSON, J. 2002. KEYNOTE SPEECH, XP 2002, SARDINIA, ITALY

[JONES97] - JONES, C. 1997. APPLIED SOFTWARE MEASUREMENTS. MCGRAW HILL.

[PARKINSON58] - PARKINSON, N. 1958. PARKINSON’S LAW: THE PURSUIT OF PROGRESS. JOHN MURRAY.

[STANDISH94] - JIM JOHNSON, ET. AL 1994. CHAOS: CHARTING THE SEAS OF INFORMATION TECHNOLOGY. PUBLISHED REPORT. THE STANDISH GROUP

[THOMAS01] - THOMAS, M. 2001. “IT PROJECTS SINK OR SWIM” BRITISH COMPUTER SOCIETY REVIEW.

CS466 – SOFTWARE PROCESS

AGILE & ITERATIVE DEVELOPMENT (CHAPTER 6)

A MANAGER’S GUIDE BY: CRAIG LARMAN

WEEK 6: EVIDENCE

BY: JOSEPH MARTINAZZI

EVIDENCE

WHAT ARE THE MOST EXCITING, PROMISING SOFTWARE ENGINEERING IDEAS OR TECHNIQUES ON THE HORIZON?

I DON'T THINK THAT THE MOST PROMISING IDEAS ARE ON THE HORIZON. THEY ARE ALREADY HERE AND HAVE BEEN FOR YEARS BUT ARE NOT BEING USED PROPERLY. – DAVID L. PARNAS

TODAY'S LECTURE FOCUSES ON WHY ITERATIVE AND INCREMENTAL DEVELOPMENT (IID) HAS A HIGHER PROBABILITY OF SUCCESS COMPARED TO THE WATERFALL MODEL. TOPICS THAT WILL BE COVERED INCLUDE:

1. RESEARCH EVIDENCE – STUDIES THAT PROVE PROGRAMS THAT USE AN IID APPROACH HAVE LOWER RISK, ARE MORE EFFICIENT, AND PRODUCES A HIGHER QUALITY PRODUCT.
2. EARLY LARGE PROJECT EVIDENCE – EXAMPLES OF LIFE-CRITICAL SYSTEMS THAT HAVE SUCCESSFULLY BEEN DEVELOPED USING AN IID APPROACH.
3. STANDARDS-BODY EVIDENCE – DESCRIBES HOW THE DOD ADOPTED MIL-STD-498 IN 1987 THAT UTILIZES ITERATIVE AND EVOLUTIONARY METHODS.
4. EXPERT THOUGHT LEADER EVIDENCE – EXAMPLES OF PROMINENT SOFTWARE ENGINEERS AND THEIR RECOMMENDATION TO ADOPT AN IID APPROACH TO SOFTWARE DEVELOPMENT.
5. A BUSINESS CASE – A COMPARISON OF AN IID APPROACH TO SOFTWARE DEVELOPMENT VS. A SERIAL WATERFALL APPROACH.
6. WATER FALL PROBLEMS AND WHY IT IS STILL PROMOTED – COMPANIES LIKE THE IDEA OF "REQUIREMENT DEVELOPMENT IS COMPLETE" PRIOR TO BEGINNING SOFTWARE DEVELOPMENT.

1. RESEARCH EVIDENCE

THE AUTHOR POINTS TO VARIOUS STUDIES THAT SHOW EVOLUTIONARY DEVELOPMENT RESULTS IN A HIGHER PROBABILITY OF SUCCESS COMPARED TO PROGRAMS THAT FOLLOW A WATERFALL MODEL.

A STUDY LEAD BY ALAN MAC CORMACK [MACCORMACK01] IDENTIFIED 4 PRACTICES THAT WERE COMMON ACROSS THE MOST SUCCESSFUL PROGRAMS. THESE PROGRAMS:

1. FOLLOWED AN IID PROCESS WHICH EMPHASIZED AN EARLY RELEASE OF THE PRODUCT TO THE STAKEHOLDERS FOR REVIEW AND FEEDBACK. [COMMON AMONG ALL IID] – *"SOFTWARE DEVELOPMENT BEST PRACTICE"*
2. DAILY INCORPORATION OF NEW SOFTWARE ONTO A REGRESSION TESTED BUILD. [COMMON AMONG ALL IID]
3. A TEAM EXPERIENCED IN SHIPPING MULTIPLE PROJECTS.
4. EARLY ATTENTION TO SYSTEM ARCHITECTURE AND COUPLING OF MAJOR COMPONENTS [UP PRACTICE]

DEFECT DENSITY IS THE NUMBER OF DEFECTS FOUND IN THE SOFTWARE/MODULE DURING A SPECIFIC PERIOD OF OPERATION OR DEVELOPMENT DIVIDED BY THE SIZE OF THE SOFTWARE/MODULE. IT ENABLES ONE TO DECIDE IF A PIECE OF SOFTWARE IS READY TO BE RELEASED. DEFECT DENSITY IS COUNTED PER THOUSAND LINES OF CODE ALSO KNOWN AS KLOC.

1. RESEARCH EVIDENCE

A FOLLOW-UP STUDY LEAD BY MAC CORMACK [MKCC03] IDENTIFIED 2 DRIVING IID FACTORS THAT IMPACTED DEFECT DENSITY.

- BY RELEASING THE SYSTEM EARLY (E.G. WHEN 20% OF THE FUNCTIONALITY WAS COMPLETE VS. 40%), THE ESCAPING DEFECT RATE DECREASED BY 10 DEFECTS/MONTH PER MILLION LINES OF CODE.
- BY CONTINUOUSLY INTEGRATING CODE ONTO A REGRESSION TESTED DAILY BUILD, THE ESCAPING DEFECT RATE DECREASED BY 13 DEFECTS/MONTH PER MILLION LINES OF CODE.

  THIS IMPLIES THAT MORE IN-PHASE DEFECTS WERE DETECTED DURING CODE/UNIT TEST MAKING THEM CHEAPER TO FIX!

  (ON PAGE 79 OF THE TEXT, THE AUTHOR STATES SEVERAL CASE STUDIES REPORT LOWER DEFECT DENSITIES ARE ASSOCIATED WITH IID METHODS [MANZO02], HOWEVER THEY ARE NOT STATISTICALLY RELIABLE.)

THE SAME STUDY ALSO IDENTIFIED THAT THESE SAME FACTORS IMPACTED PRODUCTIVITY.

- BY RELEASING THE PRODUCT EARLY (E.G. WHEN 20% OF THE FUNCTIONALITY WAS COMPLETE VS. 40%), 8 ADDITIONAL LINES OF SOURCE CODE WERE PRODUCED BY EACH PERSON DAILY.
- BY CONTINUOUSLY INTEGRATING CODE ONTO A REGRESSION TESTED DAILY BUILD, 17 ADDITIONAL LINES OF SOURCE CODE WERE PRODUCED BY EACH PERSON DAILY.

1. RESEARCH EVIDENCE

ANOTHER LARGE STUDY CONDUCTED BY THE STANDISH GROUP [STANDISH98] ANALYZED 23,000 PROJECTS. THIS STUDY FOUND THAT 4 OF THE TOP 5 FACTORS IN SUCCESSFUL PROJECTS WERE RELATED TO IID METHODOLOGIES.

- HIGH USER INVOLVEMENT – WITH SHORT ITERATIONS, DEMOS, REVIEWS, EVOLUTIONARY REQUIREMENT REFINEMENT, AND CLIENT DRIVEN ITERATIONS
- EXECUTIVE SUPPORT – FOCUSED ON TANGIBLE RESULTS
- CLEAR BUSINESS OBJECTIVES – DRIVEN BY CLIENT-DRIVEN PLANNING
- EXPERIENCED PROJECT MANAGER
- SMALL MILESTONES – ARE AT THE HEART OF THE IID METHODOLOGY

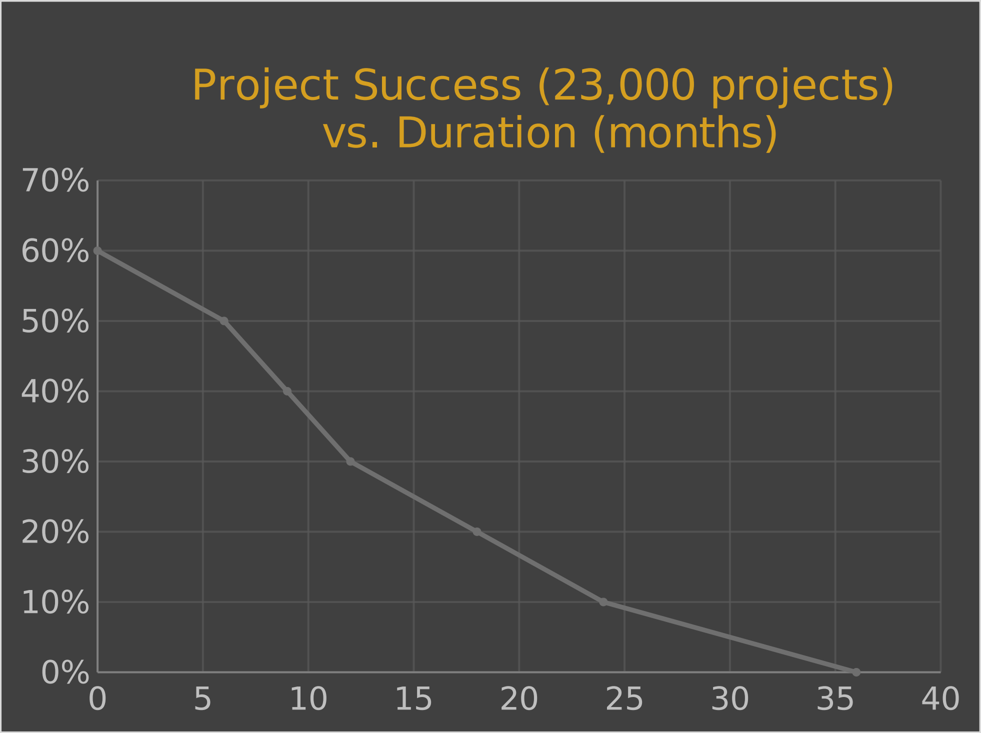
1. RESEARCH EVIDENCE – *SIZE RESEARCH*

THIS SAME STUDY [STANDISH98] ALSO ANALYZED PROJECT SUCCESS, BASED ON THE PROJECT COMPLETING WITHIN COST/SCHEDULE AND CONTAINING ALL THE SPECIFIED FUNCTIONALITY, IN RELATIONSHIP TO DURATION. AS SHOWN IN THE GRAPH TO THE LEFT; SMALLER PROJECTS THAT COMPLETED IN SEVERAL MONTHS EXPERIENCED A HIGHER SUCCESS RATE THAN LARGER PROJECTS LASTING 36 MONTHS.

- SMALL PROJECTS ARE LESS COMPLEX AND TAKE LESS TIME TO COMPLETE.
- FOR A LARGE PROJECT TO BE SUCCESSFUL, IT MUST BE BROKEN DOWN INTO SMALL (LESS COMPLEX) ITERATIONS.

THIS TREND WAS CONFIRMED BY A FOLLOW-UP STUDY SPANNING 35,000 PROJECTS [STANDISH00] THAT FOCUSED ON COST.

- SMALL PROJECTS ARE LESS COSTLY TO COMPLETE.
- FOR A LARGE PROJECT TO BE SUCCESSFUL, IT MUST BE BROKEN DOWN INTO SMALL (LESS COMPLEX) ITERATIONS.


Project Success (23,000 projects) vs. Duration (months)

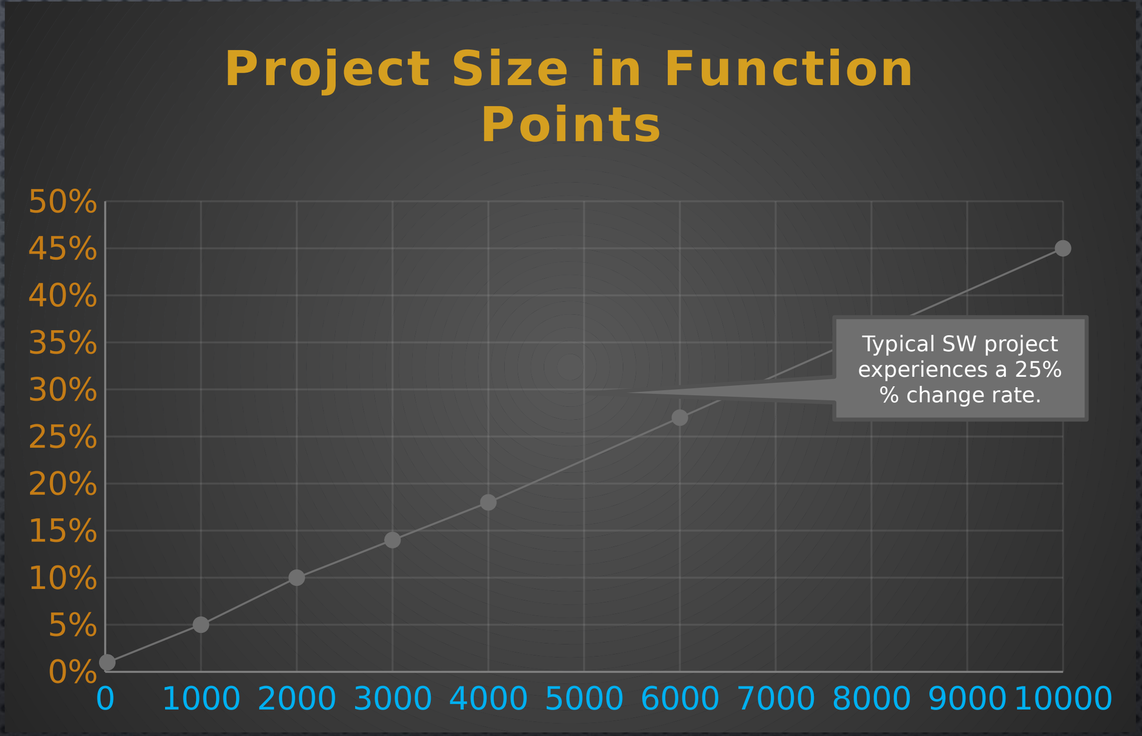
Cost	<0.5 M	0.5M-3M	3M - 6M	6M-10M	>10 M
Success	68%	22%	9%	1%	0%

UP, XP, & SCRUM

1. RESEARCH EVIDENCE – *CHANGE RESEARCH*

THE FOLLOWING GRAPH IS BASED ON RESULTS FROM MULTIPLE LARGE-SCALE SOFTWARE DEVELOPMENT PROJECTS. [JONES97]

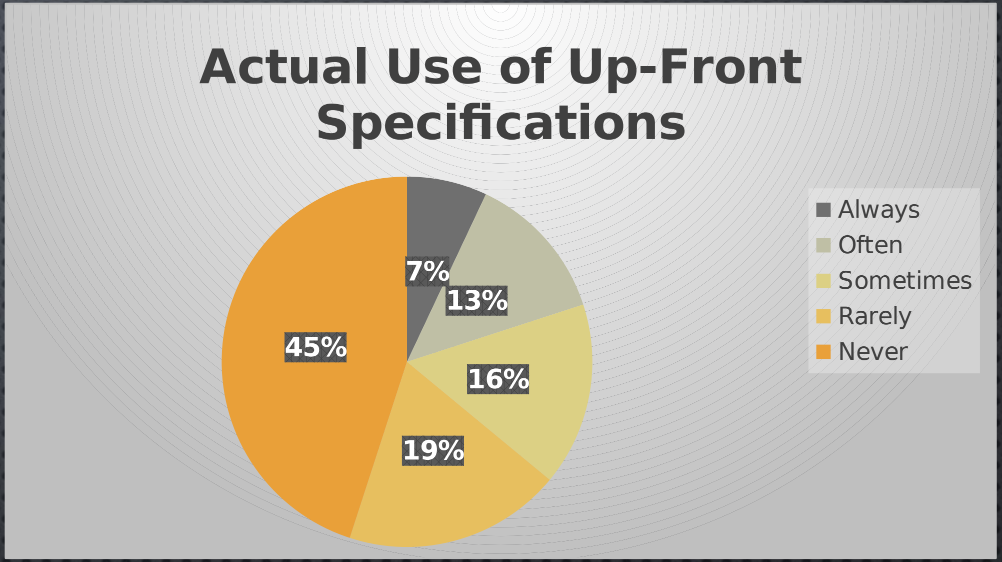
- IT ILLUSTRATES THAT AS THE COMPLEXITY OF THE PROJECT INCREASES (FUNCTION POINTS) THE AMOUNT OF REQUIREMENT CHANGE (OR CREEP) ALSO INCREASES.
- MEDIUM SIZE PROJECTS HAVE A CHANGE RATE OF 25%
- LARGE SIZE PROJECTS HAVE A CHANGE RATE OF 35%



THIS ENFORCES THE CONCEPT THAT AN ITERATIVE LIFECYCLE MODEL HAS A BETTER CHANCE OF SUCCESS THAN A SEQUENTIAL LIFECYCLE MODEL SINCE IT CAN ADAPT BETTER TO CHANGING REQUIREMENTS, FOCUSES ON ARCHITECTURE AND HIGH-RISK REQUIREMENTS EARLY, HAS A BETTER PRODUCTIVITY RATE, AND A PRODUCES A HIGHER QUALITY PRODUCT (ONE WITH FEWER DEFECTS).

1. RESEARCH EVIDENCE – *CHANGE RESEARCH*

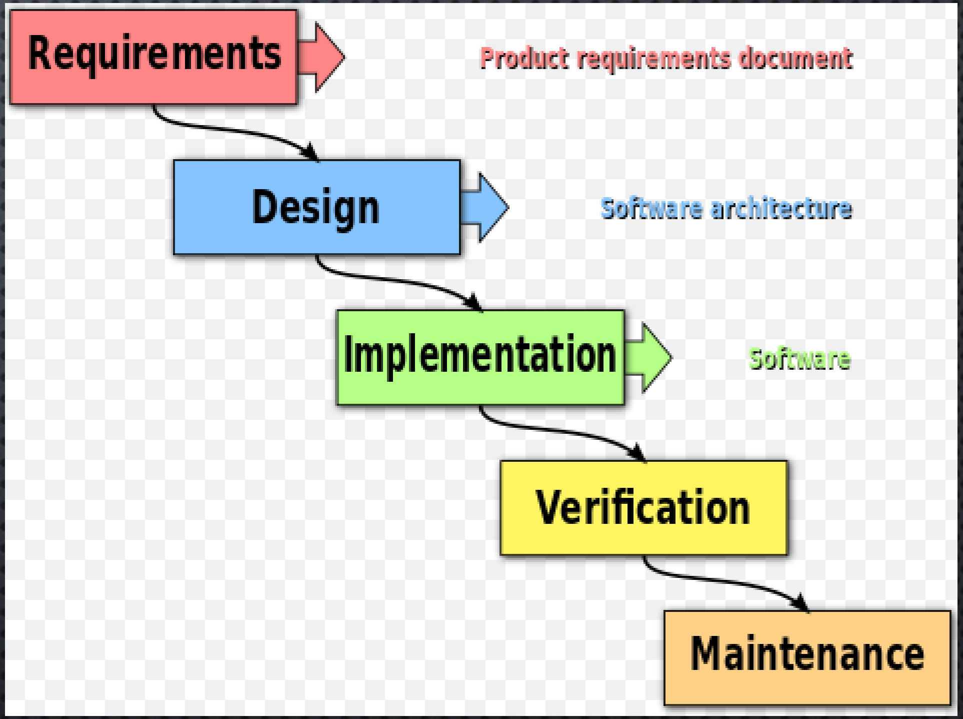
- IN ANOTHER STUDY THE AUTHOR STATES THAT UP-FRONT SPECIFICATION WITH A SIGN-OFF CAN NOT BE SUCCESSFULLY CREATED AND THAT A STUDY SHOWED THAT 45% OF THE FEATURES CREATED FROM EARLY SPECIFICATION WERE NEVER USED, WITH AN ADDITIONAL 19% RARELY USED [JOHNSON02]
- THE AUTHOR THEN PROCEEDED TO SAY "AVOID PREDICTIVE PLANNING BECAUSE YOU CAN NOT SIMPLY PLAN THE WORK AND WORK THE PLAN" WHEN DOING ITERATIVE SOFTWARE DEVELOPMENT.
- THIS WILL ONLY WORK IF YOU PROJECT IS NOT FIRM FIXED PRICE OR IF YOUR CUSTOMER HAS BOUGHT INTO THE IDEA OF YOU DELIVERING A SYSTEM WITH ONLY 75%-95% OF THE FEATURES THEY CONTRACTED!



1. RESEARCH EVIDENCE – *WATERFALL FAILURE RESEARCH*

THE AUTHOR PROVIDED NUMEROUS EXAMPLES OF STUDIES SHOWING HOW MOST PROGRAMS FOLLOWING THE WATERFALL LIFE-CYCLE MODELED FAILED.

1. IN A STUDY OF 1,027 IT PROJECTS IN THE UK [THOMAS01]THAT USED THE WATERFALL METHODOLOGY; 87% OF THE PROJECTS FAILED. OF THESE FAILED PROJECTS, 82% CITED THE NUMBER ONE PROBLEM WAS DEVELOPING ALL THE REQUIREMENTS UP FRONT.
2. PREVIOUSLY THE DEPARTMENT OF DEFENSE (DOD) REQUIRED PROJECTS TO ADHERE TO STANDARD DOD-STD-2167 WHICH REQUIRED THE USE OF THE WATERFALL LIFECYCLE MODEL. THIS RESULTED IN 75% OF THE DOD PROJECTS FAILING OR NEVER BEING USED.
3. ONE STUDY [JARZOMBEK99] FOUND THAT EVEN THOUGH 46% OF THE SYSTEMS DEVELOPED FOR THE DOD MET THE SPECIFICATIONS, THEY FAILED TO MEET THE REAL NEEDS OF THE CUSTOMER AND WERE NEVER SUCCESSFULLY USED.
4. ANOTHER STUDY IDENTIFIED THAT THE INABILITY TO DEAL WITH CHANGING REQUIREMENTS AND LATE INTEGRATION WERE ALSO SIGNIFICANT CONTRIBUTORS TO FAILED PROJECTS [JONES95].



Wikibooks Creative Commons

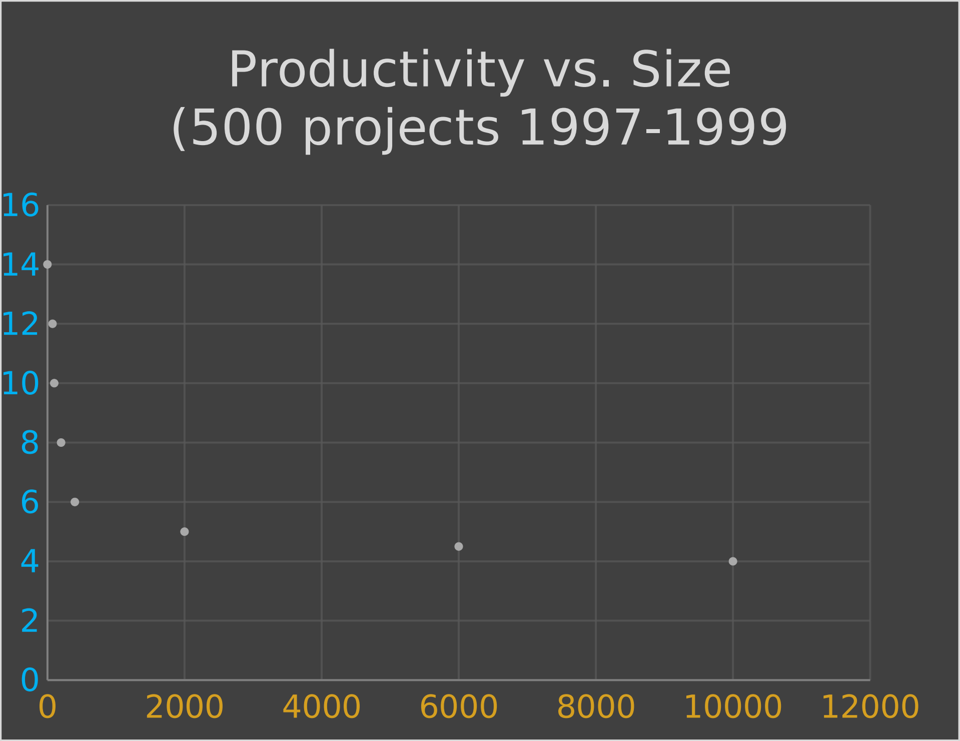
1. RESEARCH EVIDENCE – *PRODUCTIVITY RESEARCH*

- IN A STUDY [JONES00] COMPARED 500 PROJECTS FROM 1997-1999 AND FOUND THAT AS THE SIZE OF FUNCTION POINTS IN A PROJECT INCREASES, THE MONTHLY PRODUCTIVITY OF THE STAFF DECREASES.

  THIS MEANS THAT PROJECTS WITH 1,000 OR FEWER FUNCTION POINTS ARE THE MOST PRODUCTIVE AS SHOWN IN THE GRAPH TO THE LEFT.

- IN A STUDY[MARTIN91] FOUND THAT TIMEBOXING ITERATIONS ALSO SIGNIFICANTLY INCREASED PRODUCTIVITY.

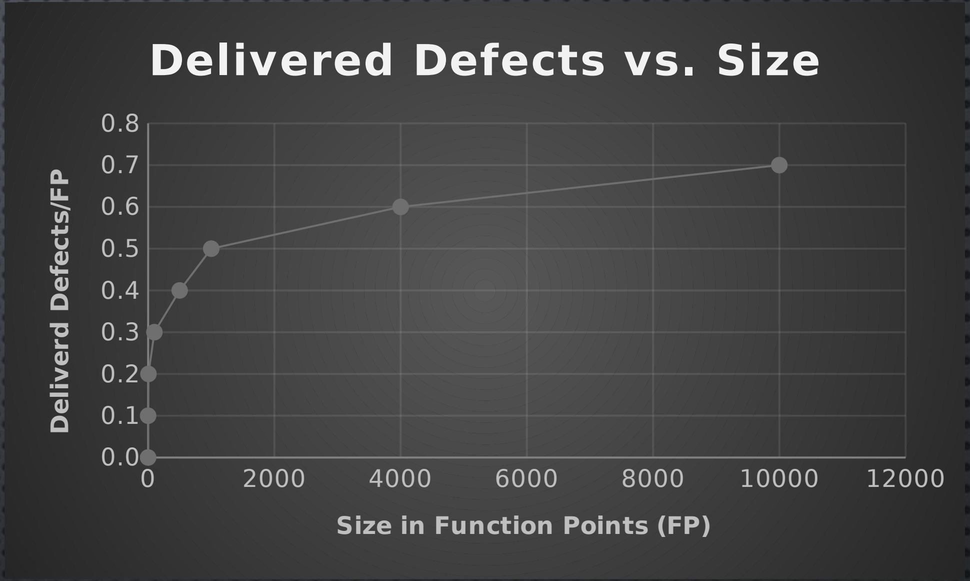
- IN ANOTHER STUDY [JONES00] FOUND THAT PRODUCTIVITY IS ALSO IMPACTED BY COMPLEXITY AS SHOWN IN THE TABLE TO THE LEFT.



	Low Complexity	High Complexity
Productivity	13%	-35%

1. RESEARCH EVIDENCE – *QUALITY & DEFECT RESEARCH*

- DEFECT REDUCTION COMES FROM AVOIDING DEFECTS BEFORE THEY OCCUR (DEMING'S TOTAL QUALITY MANAGEMENT PRINCIPAL) AND FROM FEEDBACK (PEER REVIEWS, TEST, DEMOS ETC.)
- THE AUTHOR POINT TO VARIOUS STUDIES TO SHOW THE BENEFITS OF IID:
  - [MKCC03] INDICATING IID WAS CORRELATED TO LOWER DEFECTS,
  - [MV101] INDICATING THAT DUE TO LESS TIME BETWEEN CODING AND TESTING, DEFECT RATES DECREASE,
  - [DECK94] SHOWS A STATISTICALLY SIGNIFICANT REDUCTION IN DEFECTS USING AN IID APPROACH.
- IID METHODOLOGIES:
  - ENCOURAGE CONTINUOUS PROCESS IMPROVEMENT BY MEASURING, REFLECTING, AND ADJUSTING EACH ITERATION.
  - EMPHASIZES EARLY DEVELOPMENT OF RISKY ITEMS, DEMOS, AND TEST-DRIVEN DEVELOPMENT.



The author then points to a large case study by [Jones00] that show defect rates increase non-linearly as the project size grows.
Finally, the author states several case studies report lower defect densities are associated with IID methods [Manzo02], *however they are NOT statistically reliable!*

2. EARLY HISTORICAL PROJECT EVIDENCE

PRE-1970:

1958 PROJECT MERCURY – USED ITERATIVE DEVELOPMENT TO SUCCESSFULLY BUILD THE SYSTEM INCREMENTALLY.

1970S:

THIS PROJECT LAID THE FOUNDATION FOR THE IBM FEDERAL SYSTEMS DIVISION (FSD) WHICH BUILT MANY AEROSPACE AND DEFENSE SYSTEMS THROUGHOUT THE 1970S INCLUDING:

THE US TRIDENT SUBMARINE (1972) WHICH USED 4 TIMEBOXED ITERATIONS OF 6 MONTHS IN DURATION.

THE TRW/ARMY SITE DEFENSE SOFTWARE PROJECT FOR BALLISTIC MISSILE DEFENSE WHICH DEVELOPED THE SYSTEM IN 5 ITERATIONS WITHOUT TIMEBOXING.

US NAVY HELICOPTER-SHIP SYSTEMS LAMPS THAT USED 45, 1-MONTH ITERATIONS TO SUCCESSFULLY DEVELOP THE SYSTEM.

1977-1980 - PRIMARY AVIONICS SOFTWARE SYSTEM FOR THE SPACE SHUTTLE WAS BUILT IN 17 ITERATIONS OVER 31 MONTHS AVERAGING 8 MONTHS/ITERATION.

EVERYONE OF THESE SYSTEMS WERE DEVELOPED ON TIME AND UNDER BUDGET.

2. EARLY HISTORICAL PROJECT EVIDENCE

1980S:

1984-1988 – MAGNAVOX ELECTRONIC SYSTEMS ARTILLERY COMMAND AND CONTROL SYSTEM FOR THE US ARMY WAS BUILT IN 5 ITERATIONS.

1983-1994 – US AIR TRAFFIC CONTROL (ATC) WAS RUN USING THE TRADITIONAL WATERFALL MODEL. IT FAILED DUE TO LACK OF STAKEHOLDER FEEDBACK, ANALYSIS PARALYSIS, COMPLEXITY OVERLOAD, ETC. THIS PROJECT WAS RESTARTED USING ITERATIVE DEVELOPMENT AND SUCCEEDED.

1990S:

THE CANADIAN AIR TRAFFIC CONTROL (CAATS) PROJECT IS ANOTHER EXAMPLE OF A FAILED PROGRAM THAT WAS RE-STARTED USING A UNIFIED PROCESS APPROACH WITH 6-MONTH ITERATIONS, A STAFF OF SEVERAL HUNDRED DEVELOPERS, AND OVER 1-MILLION LINES OF CODE (ADA). THE PROGRAM WAS SUCCESSFUL.

- THE PROGRAM WAS DEVELOPED BY A TEAM OF ENGINEERS THAT ORIGINATED AT HUGHES AIRCRAFT COMPANY, FULLERTON CA. AFTER WINNING THE CONTRACT THE TEAM WAS RE-LOCATED TO CANADA TO FORM HUGHES CANADA.
- THIS COMPANY WAS BOUGHT BY RAYTHEON AND TURNED INTO RAYTHEON CANADA.

3. STANDARDS-BODY EVIDENCE

TRANSITION OF US DOD STANDARDS FROM WATERFALL (1980) TO ITERATIVE AND EVOLUTIONARY (TODAY)

1980 – DOD-STD-2167 REQUIRED SOFTWARE DEVELOPMENT TO USE A WATERFALL LIFE-CYCLE MODEL AND FOLLOW A DOCUMENTATION DRIVEN APPROACH.

1988 – DOD-STD-2167A REVISED DOD-STD-2167 TO ENCOURAGE IID ALTERNATIVES TO THE WATERFALL LIFE-CYCLE MODEL. HOWEVER, SINCE THE STANDARD STILL FOCUSED ON A DOCUMENT DRIVEN APPROACH TO DEVELOPMENT, MANY CONTRACTS STILL INTERPRETED IT AS IMPLYING THEY SHOULD CONTINUE USING THE WATERFALL LIFE-CYCLE MODEL.

1994 – MIL-STD-498 SUPERSEDED DOD-STD-2167A. THIS STANDARD PROMOTED AN EVOLUTIONARY REQUIREMENTS AND DESIGN APPROACH FOR ALL INCREMENTAL ITERATIONS.

2002 THE US FOOD AND DRUG ADMINISTRATION (FDA) ALSO UPDATED THEIR STANDARDS TO ELIMINATE THE REQUIREMENT OF FOLLOWING THE WATERFALL LIFE-CYCLE MODEL AND REPLACED IT WITH ITERATIVE DEVELOPMENT.

ALTHOUGH MANY EUROPEAN STANDARDS STILL REQUIRE THE USE OF THE WATERFALL LIFE-CYCLE MODEL; NATO MADE THE LEAP TO EVOLUTIONARY DEVELOPMENT IN 2002.

4. EXPERT AND THOUGHT LEADER EVIDENCE

THE AUTHOR IDENTIFIES EXPERTS IN THE FIELD IDD INCLUDING:

HARLAN MILLS – WHO WORKED AT IBM FSD IN 1970. MILLS WAS A MAJOR CONTRIBUTOR TO THE CONCEPT OF STRUCTURED PROGRAMMING, TOP-DOWN DESIGN PROGRAMMING, AND INCREMENTAL DEVELOPMENT. MILLS STATED THAT *"SOFTWARE DEVELOPMENT SHOULD BE DONE INCREMENTALLY IN STAGES WITH CONTINUOUS USER PARTICIPATION AND REPLANNING WITH DESIGN TO COST PROGRAMMING WITHIN EACH STAGE".*

TOM GILB – PROMOTED THE "EVO" ITERATIVE METHOD IN 1976. GILB FOCUS WAS TO BREAK COMPLEX SYSTEMS DOWN INTO SMALL STEPS THAT HAD A CLEAR MEASURE OF SUCCESS. AN ADVANTAGE OF THIS APPROACH WAS THAT IF A STEP FAILED, IT GAVE YOU AN OPPORTUNITY TO INCORPORATE FEEDBACK, ADAPT, AND CONTINUE WITH THE SOFTWARE DEVELOPMENT.

FREDERICK BROOKS – RECOMMEND AN IID APPROACH TO SOFTWARE DEVELOPMENT OVER USE OF THE WATERFALL METHOD IN 1987 STATING THAT UP-FRONT REQUIREMENT SPECIFICATIONS WERE TO BLAME FOR THE HIGH PERCENTAGE OF PROGRAM FAILURES. BROOKS ALSO PUBLISHED THE BOOK TITLED – *THE MYTHICAL MAN MONTH* IN 1985 THAT STATED, *"ADDING MANPOWER TO A LATE SOFTWARE PROJECT MAKES IT LATER".*

BARRY BOEHM – PUBLISHED THE BOEHM'S SPIRAL MODEL THAT PROMOTED ITERATIVE DEVELOPMENT IN 1985.

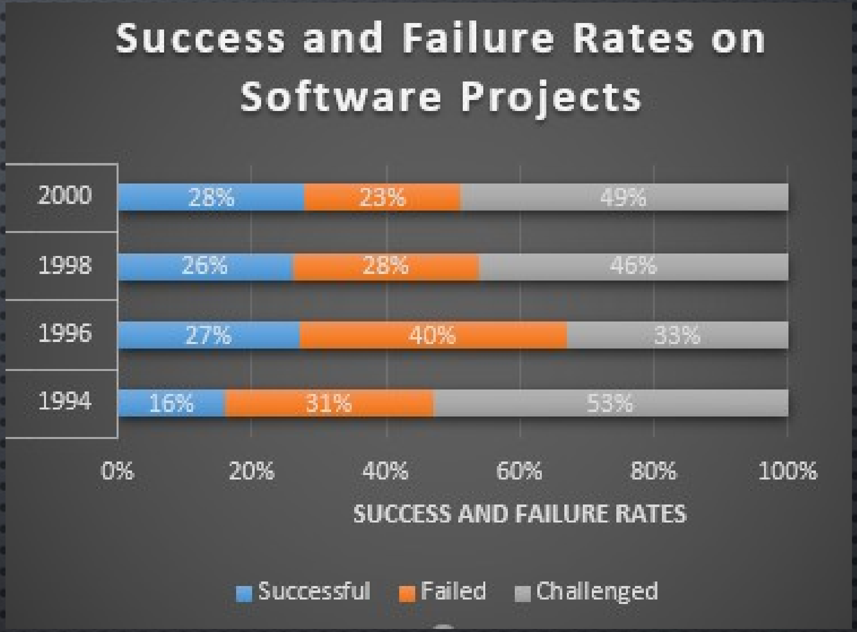
JAMES MARTIN – PROMOTED TIMEBOXED ITERATIVE DEVELOPMENT WITH CUSTOMER PARTICIPATION IN THE 1980S. MARTIN BELIEVED THAT RAPID APPLICATION DEVELOPMENT (RAD) WAS A METHOD TO UNDERSTAND LARGE COMPLEX SYSTEMS. THIS METHOD FOCUSED ON CREATING A PRODUCTION-GRADE PROTOTYPE, LEARNING FROM IT, AND EVOLVING IT UNTIL IT PRODUCED A PRODUCT THAT THE END USER WANTED.

TOM DEMARCO – IDENTIFIED IID METHODOLOGY AS A RISK MITIGATION TECHNIQUE IN 2003.

5. A BUSINESS CASE FOR ITERATIVE DEVELOPMENT

A BUSINESS CASE FOR ITERATIVE DEVELOPMENT CAN BE MADE BASED ON SEVERAL FACTORS INCLUDING:

- PRODUCTIVITY (INCREASES)
- QUALITY (PROCESS AND PRODUCT)
- LESS FAILURES, LESS COST/SCHEDULE IMPACT
- CUSTOMER SATISFACTION (END-PRODUCT MEETS EXPECTATIONS)



BASED ON THE GRAPH TO THE LEFT, IF A COMPANY AVERAGED 10 PROJECTS A YEAR AND EACH PROJECT COST $1M. THEN IN THE 2000 A COMPANY WOULD STAND TO LOOSE 23% OR $2.3M FROM FAILED PROJECTS AND HAVE COST OVERRUNS IN 49% OF THE OTHER PROJECTS.

BY ADOPTING IID METHODOLOGIES BOTH PROJECT FAILURE RATES AND CHALLENGED RATES WOULD BE REDUCED, THEREBY INCREASING THE COMPANY'S PROFITABILITY.

6. THE HISTORICAL ACCIDENT OF WATERFALL VALIDITY

WINSTON ROYCE - PUBLISHED A PAPER IN 1970 TITLED *"MANAGING THE DEVELOPMENT OF LARGE SOFTWARE SYSTEMS (LSS)"* THAT RECOMMEND TO DO THE WATERFALL PROCESS TWICE. TO FIRST HAVE A THROW-AWAY PROTOTYPE EFFORT PRIOR TO IMPLEMENTING THE PROJECT WHEN THERE ARE UNKNOWN FACTORS.

DOD-STD-2167 WAS ADOPTED IN THE 1980S WHICH REQUIRED THE USE OF THE WATERFALL MODEL COMBINED WITH DOCUMENT-DRIVEN REVIEWS. MOST IMPLEMENTORS OF LSS LOST SIGHT OF THE NEED TO PROTOTYPE WHEN UNKNOWN FACTORS ARE INVOLVED.

MANY OTHER STANDARDS WERE BASED ON DOD-STD-2167

WATERFALL WAS SIMPLE: DO REQUIREMENTS, DESIGN, AND IMPLEMENTATION.

WATERFALL GAVE THE ILLUSION OF AN ORDERLY, PREDICTABLE, ACCOUNTABLE, AND MEASURABLE PROCESS WITH SIMPLE DOCUMENT DRIVEN MILESTONES.

UP-FRONT SPECIFICATIONS WERE PROMOTED BY SYSTEM ENGINEERING ORGANIZATIONS.

CMMI INFLUENCED ORGANIZATIONS TO FOLLOW A DOCUMENT DRIVEN DEVELOPMENT WHICH WAS IN LINE WITH A WATER FALL METHODOLOGY.

WordPress.com - Creative Commons

REFERENCES

AGILE & ITERATIVE DEVELOPMENT, A MANAGER'S GUIDE, CRAIG LARMAN, EIGHTH EDITION, ADDISON WESLEY, NEW YORK, NY, COPYRIGHT 2004 BY PEARSON EDUCATION , INC.

SPECIFIC SOURCES THE AUTHOR QUOTED:

[DECK94] – DECK, M. 1994. "CLEANROOM SOFTWARE ENGINEERING: QUALITY IMPROVEMENT AND COST REDUCTION." *PROCEEDINGS, 12TH PACIFIC NORTHWEST SOFTWARE QUALITY CONFERENCE.*

[JARZOMBEK99] – JARZOMBEK, J 1999. *THE 5TH ANNUAL JAWS S3 PROCEEDINGS.*

[JOHNSON02] - JOHNSON, J. 2002. KEYNOTE SPEECH, XP 2002, SARDINIA, ITALY

[JONES00] – JONES, C. 2000. *SOFTWARE ASSESSMENTS, BENCHMARKS, AND BEST PRACTICES*. ADDISION-WESLEY.

[JONES95] –JONES, C. 1995. *PATTERNS OF SOFTWARE FAILURE AND SUCCESS.* INTERNATIONAL THOMPSON PRESS.

[JONES97] - JONES, C. 1997. *APPLIED SOFTWARE MEASUREMENTS*. MCGRAW HILL.

REFERENCES

SPECIFIC SOURCES THE AUTHOR QUOTED:

[MACCORMACK01] – MAC CORMACK, A. 2001. PRODUCT-DEVELOPMENT PRACTICES THAT WORK." *MIT SLOAN MANAGEMENT REVIEW*. 42(2)

[MANZO02] – MANZO, H, 2002. "ODYSSEY AND OTHER CODE SCIENCE SUCCESS STORIES." CROSSTALK: THE JOURNAL OF DEFENSE SOFTWARE ENGINEERING, OCT. 2002, USA DOD.

[MARTIN91] – MARTIN, J. 1991. RAPID APPLICATION DEVELOPMENT. MACMILLAIN

[MKCC03]- MAC CORMACK, A. KEMERER, C., CUSUMANO, M., AND CRANDALL, B. 2003. "EXPLORING TRADE-OFFS BETWEEN PRODUCTIVITY & QUALITY IN SELECTION OF SOFTWARE DEVELOPMENT PRACTICES." WORKING DRAFT SUBMITTED TO IEEE SOFTWARE.

[MV101] – MAC CORMACK. A., VERGANTI, R., AND IANSITI, M. 2001. "DEVELOPING PRODUCTS ON INTERNET TIME: THE ANATOMY OF A FLEXIBLE DEVELOPMENT PROCESS." MANAGEMENT SCIENCE. JAN 2001.

[STANDISH 98] - JIM JOHNSON, ET. AL 1998. CHAOS: A RECIPE FOR SUCCESS, 1998. PUBLISHED REPORT. THE STANDISH GROUP

[THOMAS01] - THOMAS, M. 2001. "IT PROJECTS SINK OR SWIM" BRITISH COMPUTER SOCIETY REVIEW.