Software Development Lifecycle

Steve Macbeth
Group Program Manager
Search Technology Center
Microsoft Research Asia

About Me

- Currently manage a team of 10 Program Managers at Microsoft Research Asia
- Over 20 years experience in all aspects and stages of software development and the software business
- Worked at Microsoft for 5 years
- Started two software/technology companies before joining Microsoft

Agenda

- Overview of Software Development Lifecycle
- Organization and Roles
- Break (10 minutes)
- Tools of the Trade
- Best Practices
- Break (15 minutes)
- Q&A

Definition

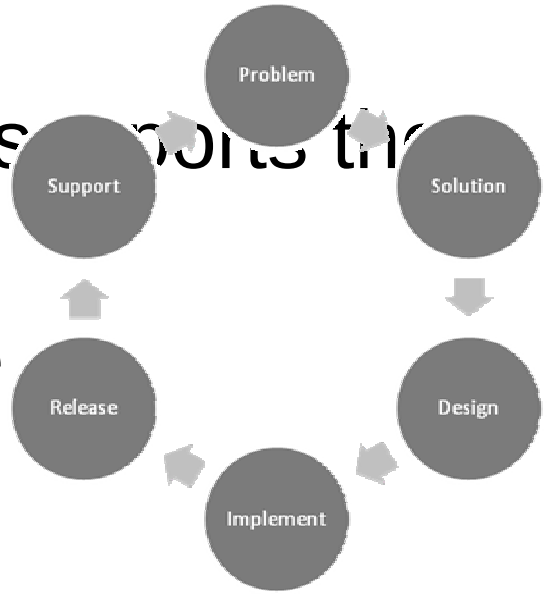
*A **software development lifecycle** is a structure imposed on the development of a software product. Synonyms include **development lifecycle** and **software process**. There are several models for such processes, each describing approaches to a variety of tasks or activities that take place during the process.*

Various Methodologies

- Waterfall
  - Traditional sequential development requirements, design, code, test, release, often used for large-scale mission critical applications
- Iterative Development
  - Small design, code, test cycles to uncover problems early, often used for commercial development contracts
- Agile Software Development
  - Built on iterative model, more people centric, relies on feedback for control, difficult to do long-term planning
- Extreme Programming
  - Built on iterative model, coding is done in pairs, design and coding are merged,
- Test Driven Development
  - Write unit test automation first, then write production code until unit test passes
- Formal Methods
  - Mathematically based, designed to ensure quality in mission critical systems

Product Development

- Identify a **Problem** that needs to be solved
- Create a plan for your **Solution** to the Problem
- **Design** the software necessary for the Solution
- **Implement** the software that supports the Design
- **Release/Deploy** the software
- **Support** the software

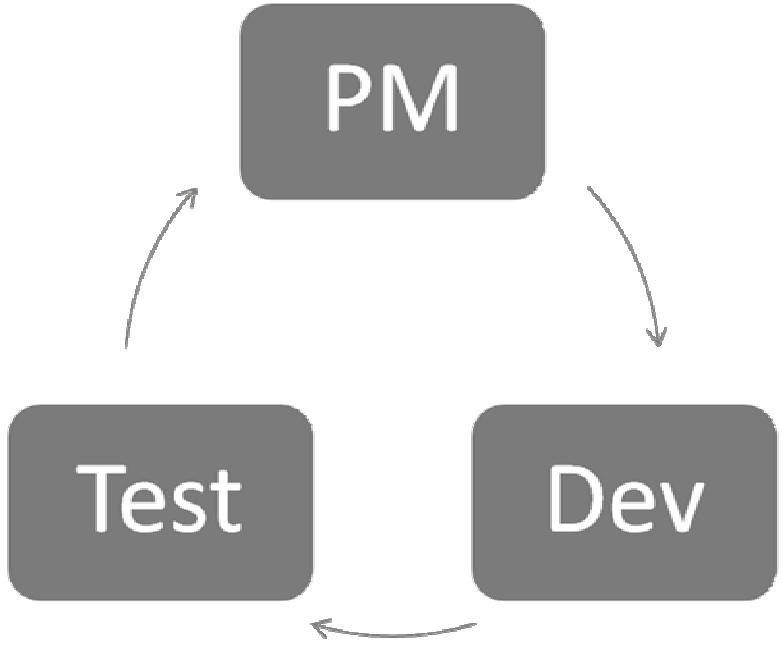


Process at Microsoft

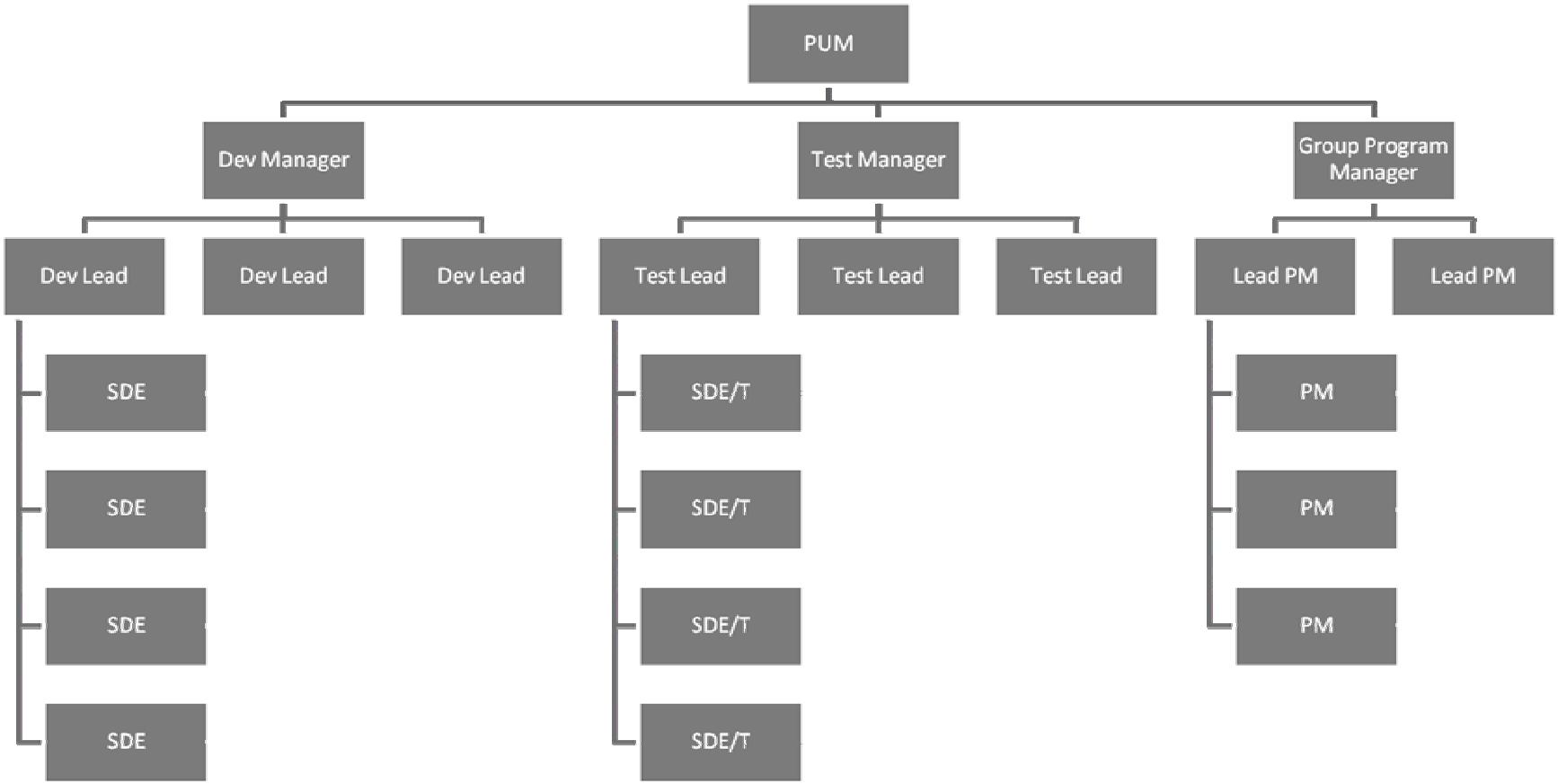
- High-level guidelines, interpreted and implemented differently across teams and projects
- Phases can overlap and can have smaller cycles nested
- Most projects use a hybrid model that is waterfall for high-level planning and release, but iterative for design/development

Engineering Disciplines

- Program Management (PM)
- Development (Dev)
- Testing (Test)



PUM Organizational Model

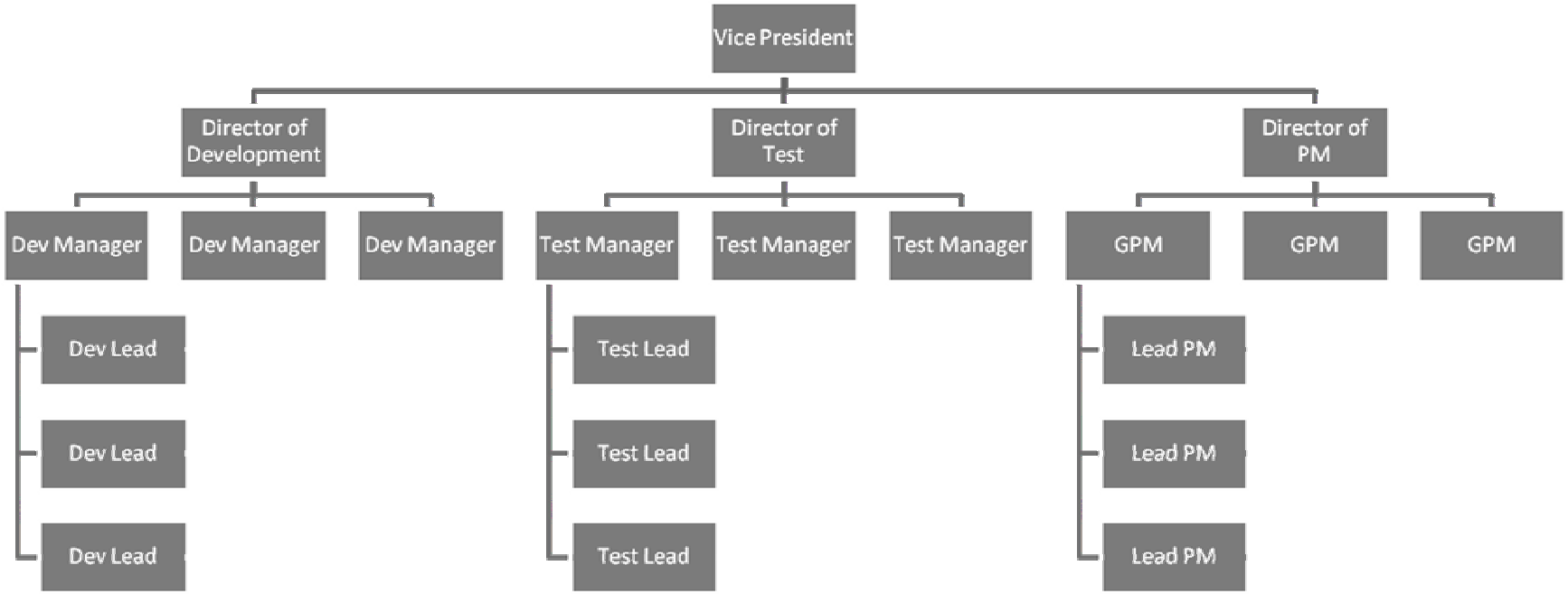


PUM Organizational Model

- Single point of ownership across disciplines
- Doesn’t scale as well to large complex systems
- Smaller teams create less career development opportunities

- Most often found in smaller teams

Functional Organizational Model



Functional Organizational Model

- Dev/Test/PM work together as a triad to make product decisions, escalation to VP for issues
- No single point of ownership on a specific feature
- Scales better to large organizations
- Creates significant critical mass within a discipline

- Most often used in large complex projects (Office, Windows, Live Services)

PM Responsibility

- Defining the **Problem Space**
  - Understanding customer requirements, industry direction, competitors
- Create **Solution Framework** together with engineering team
  - UI guidelines, system architecture, design constraints, domain modeling
- Create **Solution Specification**
  - Document high priority/impact design decisions, document exit criteria
- **Project Management**
  - Project tracking, status reporting, communication, risk assessment/mitigation

What makes a great PM?

- Someone who loves technology and is passionate about how it can be used to have a real impact on **customers** lives
- Must always be thinking about how to **optimize**
- Must be a great **diplomat**
- Must be always finding ways to **simplify**

Developer Responsibility

- Develop **Solution Framework/Specification** together with PM
- Document technical **Design/Architecture**
- Delivering quality **Code** that matches both **Solution Framework** and **Specification** and has been adequately **Tested**
- **Support** code/service after release

What makes a great Developer?

- Strong background in algorithms, mathematics, computer science
- Can **design** the simplest solution that meets the current requirements, but can be easily extended to meet unexpected requirements
- Can write quality **code** that is easy to maintain, **debug** and extend
- Can stay **focused** for long periods of time and deal with all of the **details**

Test Responsibilities

- Develop **Solution Framework/Specification** together with PM, focused on **exit/release criteria**
- Document **test cases** and **tool design**
- Develop code that will **automate** assuring the code ships at the right quality level
- Develop code that will **automate** assuring the system continues to operate in production
- Determine when the “product” is **ready to ship**

What makes a great Tester?

- Passionate about making sure our systems improve the lives of our **customers**
- Excellent **problem solving/troubleshooting** skills
- Can stay very **focused** on the smallest **details** and ensure nothing is left to chance
- Good at approaching a problem from **multiple perspectives**

Break

10 minutes

Review

- Product Development Methodologies
- Development Team Roles & Organization

- Next?
  - Roles during each phase
  - Tools of the trade
  - Best Practices

Early Phase

- Know the product vision/problem space
- Fully understand and document the key user scenarios
- Learn about your customers
- Establish good relationships between disciplines and partner teams
- Design before you code
- Research technologies and educate yourself

PM during early phase

- This part of the project is driven by PM’s
- Should start before the last phase in the previous cycle ends
- PM’s should be gathering user data, requirements, feedback, etc in order to plan next set of features

- Deliverables: vision document, problem definition, high level feature list, user scenarios

Scenarios

- Scenarios are end to end from the users perspective
- Important to really understand how users will interact with the system and to understand end to end requirements/dependencies
- Scenarios should be developed with and reviewed by real users
- Scenarios should drive feature list

Automated User Feedback

- PM team should work with engineering team to build in mechanisms to provide automated user feedback
- Query Logs/Click Through Data
- SQM/Watson
- Verbose User Feedback

Feature List

- An ordered list of features that may be built during this development cycle
- Engineering team (dev/test) provide bottom up estimates for all features (week or month resolution)
- Feature list should include impact
- Primary planning document for scoping/resource allocation

Dev during early phase

- Supporting bug fixes for previous cycle
- Training and skill development for next cycle
- Research new technologies, prototyping around core technology problems for next cycle
- Stay connected to PM team during planning
- Post-mortem from last cycle

Test during early phase

- Completing last phase of previous cycle
- Training and skill development for next cycle
- Research new technologies, prototyping around core technology problems for next cycle
- Stay connected to PM team during planning
- Post-mortem from last cycle

Middle Phase

- Divided into major milestones (M1, M2, etc.)
- Each milestone is a mini-release
  - A set of features delivered on a certain date
  - Phases
    - Planning and design
    - Implementation
    - Stabilization and Integration
    - Post-Mortem

PM during middle phase

- Completing Solution Framework/Solution Specification
- Finalizing feature list
- Managing project details (status, risk, etc.)

Solution Specification

- Well articulated Problem Definition
- Document Solution Framework so everyone on the project team is making decisions in the same way
- One line description of all features
- One page spec for all features likely to be built
- Full specs for all features planned for the first milestone

Dev during middle phase

- This part of the project is driven by Dev
- Developing design documents
- Writing code, unit testing, debugging

- Deliverables:
  - Quality code!

Unit Testing

- Developers are responsible for testing their own code, to ensure that it works within local constrains and can be checked in without breaking other code
- Unit testing can and should be automated to provide regression testing for old features during changes

Code Complete

- Target date for completing all features for this milestone
- Feature should be unit tested, checked in, integrated with other code, BVT’s pass
- Shift focus from quality at a local level to quality at a global level
- Focus on stabilizing, not adding new features

Source Code Control

- Used to manage all source code necessary to build the system
- Enables version control, roll-back, merging, branching
- Automated system to build the software from source code
- Automated system to verify new code didn’t break existing functionality, regression testing

Code Reviews

- Every line of code should be reviewed by peers before declaring code complete
- Great coaching/mentoring opportunity for junior engineering staff
- Good mechanism to ensure architectural continuity
- Ensure quality at an early stage in project

Test during middle phase

- Developing automated test frameworks to ensure quality end to end functionality, system performance, scalability
- Tracking defect rates to alert team to quality problems

- Deliverables:
  - Test automation

Defect Tracking

- Necessary to track every defect that is detected after a piece of code is declared code complete
- Triage used to determine which defects to fix, which to punt, how to resolve
- Bug Jail – used to prevent quality from getting out of control

Three Disciplines, Three Tools

- Program Manager
  - Feature List, Automated User Feedback
- Developer
  - Source Code Control System, BVT
- Tester
  - Defect Tracking System

Late Phase

- End game!
- Stabilize, tightly manage any changes
- All changes are linked to defects or design change requests
- Everyone should be focused on shipping

PM during late phase

- Working with test on driving triage
- Writing specs for design change requests
- Making sure no details are overlooked
- Starting to think about next cycle

Triage

- Triage is usually driven by either senior tester or senior PM
- During the end phase of a project all defects should be reviewed by triage team
- Determine which defects should be fixed
- Determine how defects will be resolved

Not all bugs are worth fixing!

1. When this bug happens, how bad is the impact? (**Severity**)
2. How often does this bug happen? (**Frequency**)
3. How much effort would be required to fix this bug? (**Effort**)
4. What is the risk of fixing this bug? (**Risk**)

Fixing bugs is only important when the value of having the bug fixed exceeds the cost of the fixing it.
(severity + frequency) > (effort + risk)

Dev during late phase

- Fixing high priority defects
- Participating in triage
- Helping test with integration, performance, scalability testing

Test during late phase

- The part of the project is driven by Test
- Focused on measuring/tracking quality by looking at defect rates/severity
- Manage alpha, beta and dogfood releases
- Use triage to manage all changes after code complete

- Deliverables:
  - Decision to ship!

Three disciplines, Three deliverables

- Program Management
  - Problem definition, feature list, solution specification
- Development
  - Quality code that meets solution specification
- Testing
  - Deciding when to ship

Break

15 minutes

Review

- Software Development Lifecycle
- Team structure and roles
- Tools of the trade
- Best Practices

Things to insist on!

- Vision document with executive support
- End to end user scenarios for all high priority features
- Feature list with engineering estimates
- Solution specification for all high priority features
- Code complete should only be declared when unit testing and code review is complete
- All code managed by a version control system
- All defects managed by a defect management system
- Defects come before new features
- Daily build, build breaks come before everything
- Triage all changes after code complete
- Well defined release criteria
- Test automation coverage of all high priority user scenarios
- Test decides when system is ready to ship

Open Discussion