Software Development Best Practices

Part I

Best Practices

- Describe best practices in rapid development
- Result of 20 years or more experience from many developers
- Common sense to less obvious
- Excluded
  - Fundamental development practices
  - Best philosophy but not best practice
  - Best practice, maybe, but not for development speed
  - Insufficient evidence
Ratings

• Efficacy
  – Potential reduction from nominal schedule
    • None = 0%
    • Fair = 0-10%
    • Good = 10-20%
    • Very Good = 20-30%
    • Excellent = 30%+
  – Improvement in progress visibility
    • None = 0%
    • Fair = 0-25%
    • Good = 25-50%
    • Very Good = 50-75%
    • Excellent = 75%+

Ratings

• Efficacy
  – Effect on schedule risk
    • Decreased
    • No effect
    • Increased
  – Chance of first-time and long-term success
    • Poor = 0-20%
    • Fair = 20-40%
    • Good = 40-60%
    • Very Good = 60-80%
    • Excellent = 80%+
Change Board

• Approach to controlling changes in the product
  – Brings together representatives from all parties
    • Development, QA, Doc, Customer support, Marketing, etc.
  – Gives representatives authority for accepting or rejecting proposed changes
  – Raises visibility of feature creep, reduces number of uncontrolled changes, keeps all parties involved

Change Board

• Efficacy
  – Potential reduction from nominal schedule: Fair
  – Improvement in progress visibility: Fair
  – Effect on schedule risk: Decreased Risk
  – Chance of first-time success: Very Good
  – Chance of long-term success: Excellent

• Major Risks
  – Approving too few or too many changes
Daily Build and Smoke Test

• A process where the product is completely built every day and put through some basic tests to see if it “smokes” when turned on

• On a typical project there are many developers that must integrate their code
  – “Build” means the product is compiled, linked, and combined into an executable at the end of each day
  – Test is a simple one that exercises basic functionality

Time Savings of Daily Build

• Minimized integration risk
  – Integrating code from team members one of the greatest risks
  – Daily build keeps integration errors small and manageable

• Reduces risk of low quality
  – Minimal smoke testing every day helps keep quality problems from taking over

• Easier defect diagnosis
  – Easier to pinpoint why something is broken on any given day; changes since last day; incremental development

• Supports progress monitoring
  – Obvious what features are present and missing

• Improves morale
  – Boost in morale to see the product work and progress made
  – Also applies to customer relations
Using the Daily Build and Smoke Test

• Build daily
  – Or at regular intervals
  – “Heartbeat” of the project; keeps developers synchronized
  – Use automated build tools; e.g. make

• Check for broken builds
  – Fixing broken builds is top priority
  – Failure to pass smoke test is a broken build

• Smoke test daily
  – Exercise entire system end to end but not exhaustive
  – Grows from “hello world” to complex system that may even take hours to run

Using the Daily Build and Smoke Test

• Developers should smoke test before adding to the build

• Use version control tools to know what might have broken the build and be able to revert

• Create a penalty for breaking the build
  – $$ ?
  – Beeper?
  – Sucker?
  – Responsibility for build until fixed?
Risks of Daily Build

- Tendency toward premature release
  - Developers might focus on the build and skip materials needed for the final product like documentation
  - Developers might put in hacks to fix the build

Daily Build Summary

- Efficacy
  - Potential reduction from nominal schedule: Good
  - Improvement in progress visibility: Good
  - Effect on schedule risk: Decreased Risk
  - Chance of first-time success: Very Good
  - Chance of long-term success: Excellent

- Major Risks
  - Pressure to release interim versions of a program too frequently

- Major Interactions
  - Especially effective with miniature milestones
Designing for Change

• Broad practice that encompasses many practices to plan for change. Must be employed early in the lifecycle.
  – Identifying likely changes
  – Develop a change plan
  – Hide design decisions to avoid rippling through the project

Using Designing for Change

• Identify Areas Likely to Change
  – List design decisions likely to change
  – Great designers able to anticipate more kinds of possible change than average designers
  – Frequent sources:
    • Hardware dependencies
    • File formats
    • Nonstandard language features
    • Difficult design areas
    • Specific data structures
    • Business rules
    • Requirements barely excluded
    • Features for next version
Using Designing for Change

• Use Information Hiding
  – Plenty has been said about this already
  – Hide design decisions inside modules

  – One of the few theoretical techniques proven useful in practice

Using Designing for Change

• Develop a Change Plan
  – Examples:
    • Use late-binding strategies for types or data structures that may change (e.g. allocate dynamically based on sizes)
    • Use named constants instead of hard-coded literals
    • Data-driven techniques where data dictates how the program will operate instead of hard-coding
Designing for Change Summary

• Efficacy
  – Potential reduction from nominal schedule: Fair
  – Improvement in progress visibility: None
  – Effect on schedule risk: Decreased Risk
  – Chance of first-time success: Good
  – Chance of long-term success: Excellent

• Major Risks
  – Over-reliance on programming languages to solve design problems rather than on change-oriented design practices

Evolutionary Delivery

• Lifecycle model using the ideas of evolutionary prototyping. Delivers selected portions of the software earlier than would otherwise be possible, but does not necessarily deliver the final product any faster.
• Can lead to improved quality, even distribution of development and testing
Evolutionary Delivery Approach

- Going grocery shopping
  - Waterfall model: complete list for next week
  - Prototyping: no list, get what looks good
  - Evolutionary delivery: in between, start with a list then improvise as you go

Evolutionary Delivery Benefits

- Reduces risk of delivering a product the customer doesn’t want
- Makes progress visible by early and often delivery
- Reduces risk of integration by integrating early and often
- Improves morale as the project evolves in power
Evolutionary Delivery Summary

• Efficacy
  – Potential reduction from nominal schedule: Good
  – Improvement in progress visibility: Excellent
  – Effect on schedule risk: Decreased Risk
  – Chance of first-time success: Very Good
  – Chance of long-term success: Excellent

• Major Risks
  – Feature creep, diminished project control, unrealistic schedule, inefficient use of development time

Goal Setting

• Human motivation is the single, strongest contributor to productivity
  – A manager simply tells developers what is expected
  – Developers will generally work hard to achieve a goal of “shortest schedule”
  – Primary obstacle to success is an unwillingness to define a small, clear set of goals and commit to them for an entire project
Goal Setting: Goal of Shortest Schedule

• Efficacy
  – Potential reduction from nominal schedule: Very Good
  – Improvement in progress visibility: None
  – Effect on schedule risk: Increased Risk
  – Chance of first-time success: Good
  – Chance of long-term success: Very Good

• Major Risks
  – Significant loss of motivation if goals are changed

Goal Setting: Goal of Least Risk

• Efficacy
  – Potential reduction from nominal schedule: None
  – Improvement in progress visibility: Good
  – Effect on schedule risk: Decreased Risk
  – Chance of first-time success: Good
  – Chance of long-term success: Very Good

• Major Risks
  – Significant loss of motivation if goals are changed
Goal Setting: Goal of Maximum Visibility

- Efficacy
  - Potential reduction from nominal schedule: None
  - Improvement in progress visibility: Excellent
  - Effect on schedule risk: Decreased Risk
  - Chance of first-time success: Good
  - Chance of long-term success: Very Good

- Major Risks
  - Significant loss of motivation if goals are changed

Inspections

- Formal technical review
  - Participants inspect review materials before the review meeting to stimulate discovery of defects
  - Participants have roles of moderator, scribe, participant
  - Can find errors before going to testing, studies have found it more effective in total defects found and time spent per defect
  - Good tool for tracking progress
Inspections Summary

- **Efficacy**
  - Potential reduction from nominal schedule: Very Good
  - Improvement in progress visibility: Fair
  - Effect on schedule risk: Decreased Risk
  - Chance of first-time success: Good
  - Chance of long-term success: Excellent

- **Major Risks**
  - None

Lifecycle Model Selection

- Product development styles vary tremendously among different kinds of projects
- Choice of the wrong lifecycle model can result in missing tasks and inappropriate task ordering, which undercuts planning and efficiency
- Choose the appropriate lifecycle
Lifecycle Selection Summary

- **Efficacy**
  - Potential reduction from nominal schedule: Fair
  - Improvement in progress visibility: Fair
  - Effect on schedule risk: Decreased Risk
  - Chance of first-time success: Very Good
  - Chance of long-term success: Excellent

- **Major Risks**
  - Specific lifecycle models may contain certain risks

Measurement

- **Quantitative measurement of project progress**
  - Dozens of techniques, we will discuss in more detail later
    - Size, lines of code, defect rate, hours spent debugging, hours spent designing, developer or customer satisfaction surveys
  - Provides complementary information to adjust estimates, schedules, track progress

- Can have short-term motivational benefits and long-term cost, quality, and schedule benefits
Measurement Benefits

• Provides status visibility
  – Helps you and others know what your status is
• Focuses people’s activities
  – Feedback on measurement can motivate and get people to respond; e.g. reduce defect rate
  – What gets measured gets optimized
• Improves morale
  – Properly implemented, measurement can improve morale by bringing attention to problem areas
• Help set realistic expectations
  – Provides historic baseline over long-term
  – Sets stage for process improvement

What to Measure

• Cost and resource data
  – Effort by activity, phase, personnel type
  – Computer resources
  – Time
• Change and defect data
  – Defects by classification
  – Problem report status
  – Defect detection method
  – Effort to detect and correct defects
• Process data
  – Process definition, process conformance
  – Estimated time to completion
  – Milestone progress
  – Requirement changes
• Product data
  – Size, functions
  – Development milestones
  – Total effort
Measurement Risks

- Over-reliance on statistics, data accuracy
- Over-optimization of a single factor
  - If measure LOC, developers may become more verbose but decrease quality
  - If only measure defects, development might drop in favor of testing/fixing
- Measurements misused for employee evaluations
  - Lots of defects does not necessarily mean a bad developer

Measurement Summary

- Efficacy
  - Potential reduction from nominal schedule: Very Good
  - Improvement in progress visibility: Good
  - Effect on schedule risk: Decreased Risk
  - Chance of first-time success: Good
  - Chance of long-term success: Excellent
- Major Risks
  - Over-optimization of single-factor measurements
  - Misuse of measurements for employee evaluations
  - Misleading information from LOC measurements
Miniature Milestones

- Fine-grain approach to project tracking and control
  - Provides good visibility into a project’s status
  - Keys to success include
    - Overcoming resistance of people whose work will be managed with the practice, may feel like micromanagement
    - Staying true to the “miniature” nature

Miniature Milestones

- Driving to the lower 48
  - Major milestones: cities along the way
    - Might be hundreds of miles apart
  - Mini milestones: stops and landmarks much closer, perhaps 25 miles apart
    - Move to mini milestone, then make a reading to the next mini milestone, etc.
- Define set of targets
  - Targets should be met on a daily or near daily basis
  - If milestones are not met, you know the schedule isn’t realistic and will find out early on
Miniature Milestone Benefits

- Improves status visibility
  - Avoid letting developers “go dark”
    - “How’s everything going?” “OK”
    - “How’s everything going?” “Late by 6 months.”
- Can help keep people on track
  - Easy to lose sight of the big picture without short-term milestones
- Improved motivation
  - Achievement happens regularly
- Reduced schedule risk
  - Breaks large, poorly defined schedule into smaller more well-defined ones
  - Requires more planning work on behalf of manager

Using Mini Milestones

- Initiate early or in response to a crisis
  - If set up at other times, manager runs the risk of appearing draconian and over-controlling
- Have developers create their own mini milestones
  - Allows developers to remain in control and not feel micro-managed
- Keep milestones miniature
  - Achievable in 1-2 days
  - Important to be able to catch up quickly if a milestone is missed
  - Reduces number of places for unforeseen problems to hide
- Make milestones binary
  - Done or not done
Using Mini Milestones

• Make the set of milestones exhaustive
  – Must cover every task needed to release the product
  – Do not allow developers to keep list of “cleanup” tasks in their heads, easily lost
• Use for short-term but not long-term planning
• Regularly assess progress and recalibrate or replan
  – Since mini milestones are short term they need realignment often, can't plan ahead too far

Mini Milestone Side Effects

• Requires detailed, active management
• Demands additional time and effort from both management and developers
  – Tradeoff with increased visibility and control of the planning process
• Successful use prevents a project leader from losing touch with the project
  – In regular contact with each person whenever a milestone is to be done
  – Lots of incidental communication that helps with risk management, motivation, personnel issues, and other management activities
Mini Milestones Summary

• Efficacy
  – Potential reduction from nominal schedule: Fair
  – Improvement in progress visibility: Very Good
  – Effect on schedule risk: Decreased Risk
  – Chance of first-time success: Good
  – Chance of long-term success: Excellent

• Major Risks
  – Developer opposition to micro-management

• Major Interactions
  – Especially well-suited to project recovery
  – Works well with daily build and smoke test practice