Software Project Survival Guide

How to be sure your First Important Project isn't your Last

by Steve McConnell

(author of Rapid Development and Code Complete)

Dan Broyles

Department of Electrical Engineering &
Computer Science

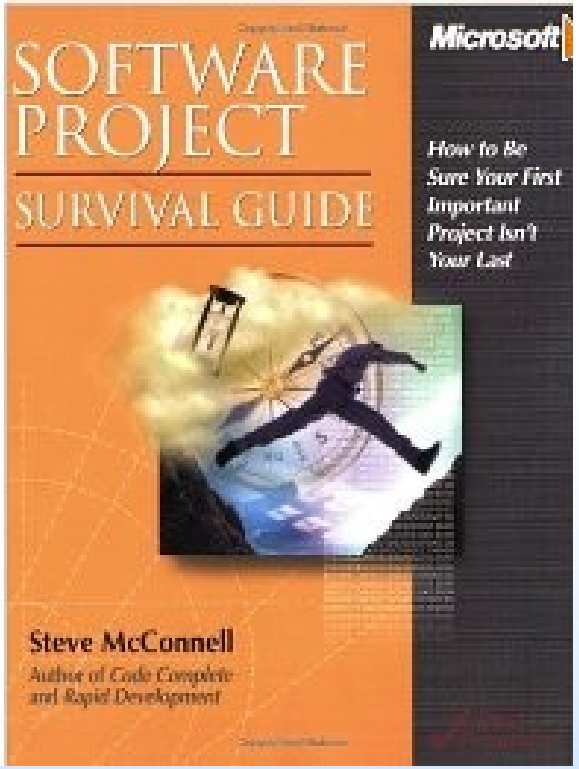
EECS 811 Class Presentation

Daniel.s.broyles@sprint.com

Software Project Survival Guide

Outline

- Introduction
- The Survival Mindset
- Survival Preparations
- Succeeding by Stages
- Mission Accomplished
- References



Software Project Survival Guide

Introduction

- Introduction
- The Survival Mindset
- Survival Preparations
- Succeeding by Stages
- Mission Accomplished
- References

Introduction
About the Author

- Steve McConnell's Publication
  - 1993: Code Complete
  - 1996: Rapid Development
  - 1998: Software Project Survival Guide
  - 1999: After the Gold Rush
  - 2004: Professional Software Development
  - 2006: Software Estimation – Demystifying the Black Art
- IEEE Software Magazine
  - 1996 – 1998: Editor of "Best Practices" Column
  - 1998 – 2002: Editor and Chief
- Created Construx Software 14 years ago to advance the art and science of commercial software engineering

Introduction
Bump, bump, bump…

Here is Edward Bear, coming downstairs now, bump, bump, bump on the back of his head, behind Christopher Robin.

It is, as far as he knows, the only way of coming downstairs, but sometimes he feels there really is another way, if only he could stop bumping for a moment and think of it.

And then he feels that perhaps there isn't.

Introduction

- Why do software projects fail?
- Book addresses most common weaknesses
- Draws from three sources
  - Key Practices of the Capability Maturity Model
  - NASA's Software Engineering Laboratory (SEL) Recommended Approach to Software Development
  - His own experiences

Software Project Survival Guide

The Survival Mindset

- Introduction
- The Survival Mindset
- Survival Preparations
- Succeeding by Stages
- Mission Accomplished
- References

The Survival Mindset
Outline

- Ch 1: Welcome to Software Project Survival Training
- Ch 2: Software Project Survival Test
- Ch 3: Survival Concepts
- Ch 4: Survival Skills
- Ch 5: The Successful Project at a Glance

The Survival Mindset
Basic Survival Needs

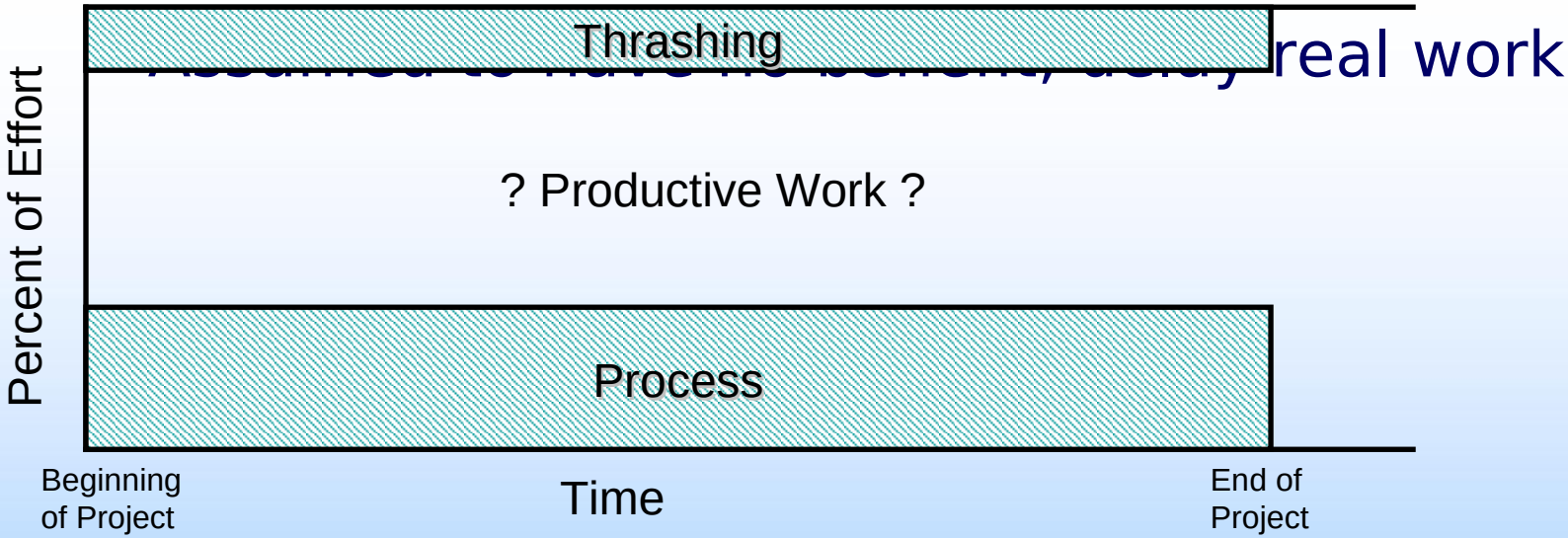
- Just like basic human needs
  - Food, air, water, security, social contact, …
- Project team needs

The Survival Mindset
Rules of Civilization

- Customer's Bill of Rights
- Project Team's Bill of Rights
- Balance - One person's rights become another person's responsibilities
- Keys to success
  - Get all parties to respect the rights
  - Satisfy each party's needs so none feel threatened

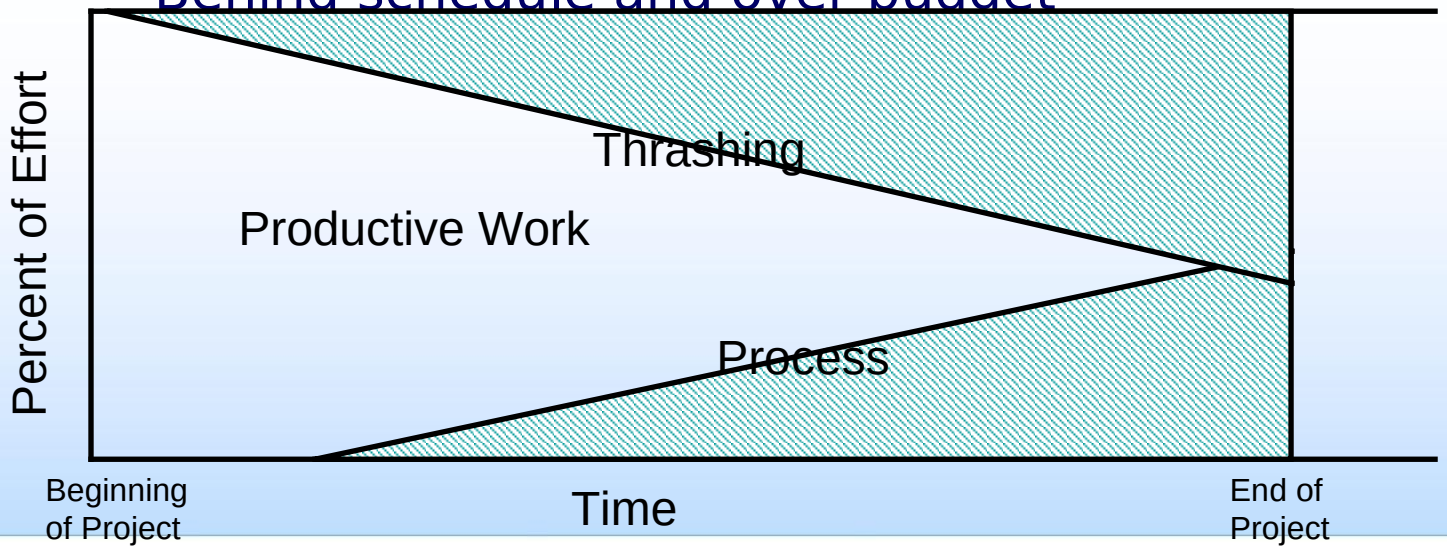
Survival Concepts
Incorrect View of Process

- "Process" is often seen as a four-letter word
  - Seen as time taken away from productive work



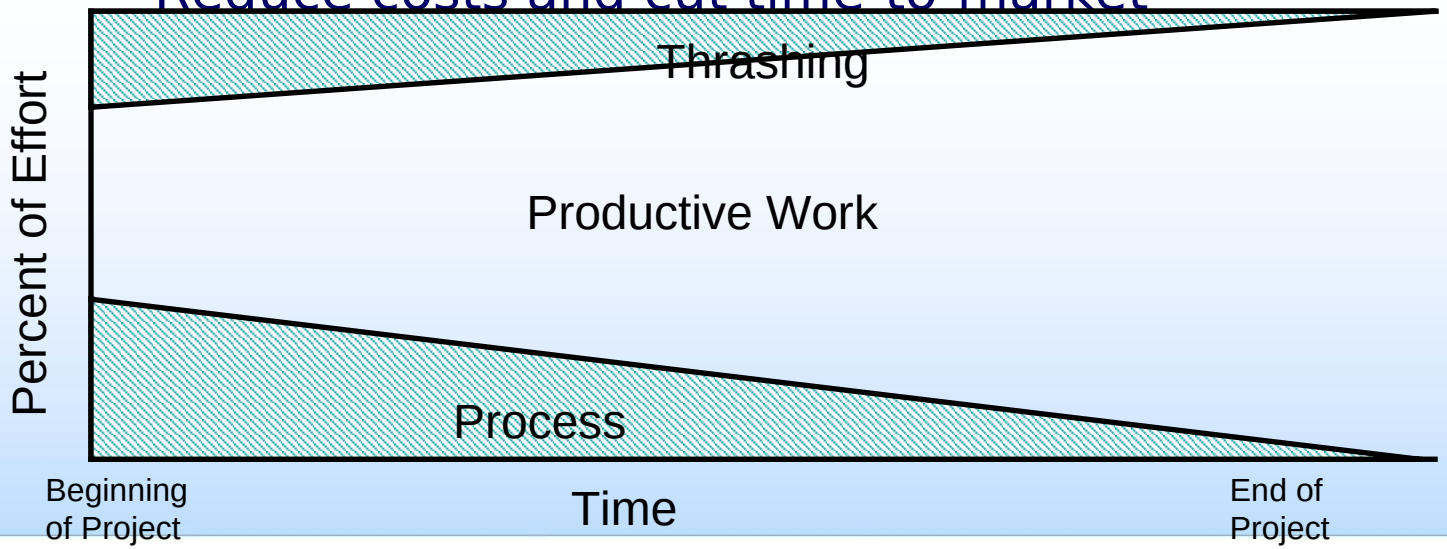
Survival Concepts
Incorrect View of Process

- Ignoring processes
  - Revisions overwhelm the project
  - Many defects and bugs, some forgotten
  - Components don't integrate well
  - Behind schedule and over budget



Survival Concepts
Process to the Rescue

- Introduce processes early in the project
  - Reduces initial productivity
  - Reduces thrashing a lot by mid-project
  - More efficiency and less process as project continues
  - Reduce costs and cut time-to-market



Survival Concepts
Process vs. Creativity and Morale

- Programmers need to be creative
- Management needs predictability, visibility, on schedule and within budget
- Is there a conflict?
- Effective processes support creativity/morale
- Programmers appreciate good leadership
  - Emphasize predictability
  - Give visibility to good work
  - Maintain control

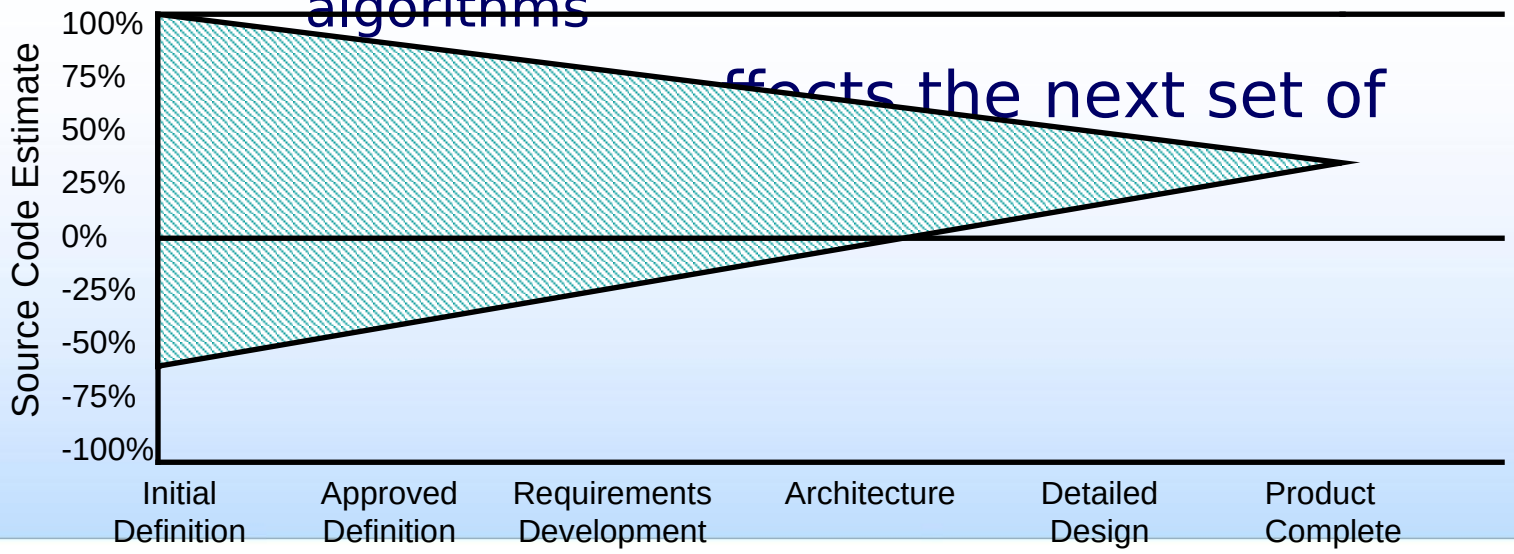
Survival Concepts
Upstream and Downstream

- Upstream
  - Early project parts: requirements and architecture
  - Good upstream work contributes to success
  - Poor upstream work can severely impair project
- Downstream
  - Later project parts like Construction and Testing
  - Upstream mistakes cost 50 to 200 times more to correct downstream than if corrected early
- Process = opportunity to fix mistakes early

Survival Concepts
Cone of Uncertainty

- Software development = refinement process
  - Large decisions lead to smaller decisions
    - Ex: Operating System -> error handling -> algorithms

ffects the next set of



Survival Concepts
Cone of Uncertainty

- Impossible to estimate project in early stages
- Scope determined by myriad of decisions
- Smart to control the way those decisions are made to meet schedule or budget targets
- Set clear, non-conflicting goals at start
- Keep product concept flexible
- Tradeoff between schedule, budget, and feature set.
  - Cannot set all three at beginning of project

Survival Skills
Systematic Approach: Planning

- Planning
  - Often overlooked by management and staff
  - Downstream benefit: 1 hour planning = 3 to 10 hours saved
- Features of a plan
  - Software development plan
  - Project estimates - Revised estimates
  - Quality assurance plan - Staged delivery plan
  - Requirements, architecture, detailed design docs

Survival Skills

Systematic Approach: Planning Checkpoint

- Planning checkpoint review
- At 10% to 20% through the project, have
  - User interface prototype
  - Detailed requirements
  - Detailed project plan with cost and schedule estimates
  - Software development and quality assurance plans

Survival Skills
Systematic Approach: Planning Checkpoint

- Two-phase funding approach
  - Request funding for first 10 to 20% of the project
  - Hold a planning checkpoint review
  - Sr. management or customer makes a “Go/No-Go” decision
  - Request funding for the remainder of the project

Survival Skills

Systematic Approach: Risk Management

- Software development is a high-risk activity
- Serious project risks are related to planning
  - Failure to plan
  - Failure to follow the plan
  - Failure to revise the plan as circumstances change
- Minimize risks with active risk management
- Practices described in book involve less risk
- Risk manager identifies top 10 project risks

Survival Skills
Systematic Approach: Project Control

- Projects can easily get out of control
- Control project to meet schedule, budget, ...
- Does not mean controlling the project team
  - Choose software life cycle model (staged delivery)
  - Manage changes in requirements
  - Set consistent design and coding standards
  - Create detailed project plan
- Good control must be planned

Survival Skills
Systematic Approach: Project Visibility

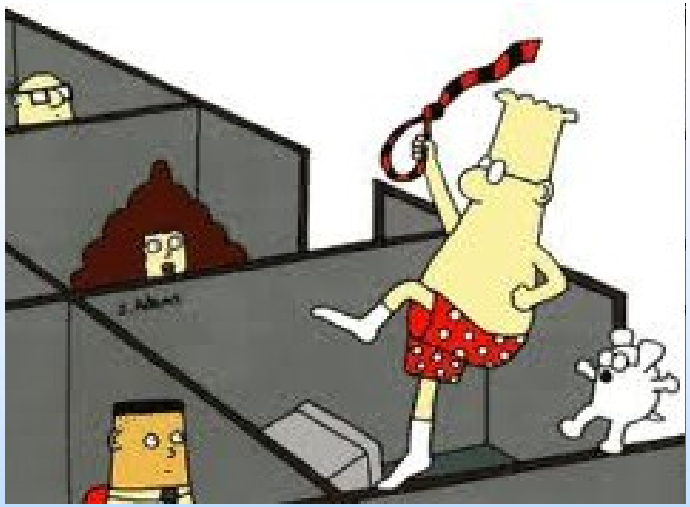
- Ability to determine project's true status
  - Use vision statement to set broad objectives
  - Hold planning checkpoint review at 10 to 20% point
  - Compare actual vs. planned performance and modify plan
  - Use binary milestones (100% "done" / "not done")
  - Compile to a releasable state regularly
  - Revise estimates at end of each phase
- Must plan visibility into project from the start
  - Does not happen automatically

Survival Skills
Systematic Approach: Peopleware

- Development requires
  - Creativity, intelligence, initiative, persistence
  - Lots of internal motivation
- Align developer's interests with assigned work
  - Motivated by jobs they find stimulating
- Developers need appreciation from sponsors
  - Not motivated by impossible goals

Survival Skills
Systematic Approach: Peopleware

- Provide thinking-oriented office space
  - Need relaxed and contemplative space
- Avoid open work bays
  - Need private space to concentrate



Survival Skills
Early User Involvement

- No real mystery to software that users love
  - Ask users what they want
  - Show users what the project team intends to build
  - Ask users how they like it
- Bring users in early while software is flexible
- Build user-oriented checkpoints into project
  - Small mid-course corrections cheaper than large corrections
  - Staged Delivery leverages this concept

Survival Skills
Product Minimalism

- Keep project simple as possible
  - Less is more
  - Avoid unnecessary complexity
  - Simplest methods are least prone to error
- Use 2-hour, 2-day, 2-week, 2-month approach
- Remove elements to make software simpler
- Adding elements make it more complex

Survival Skills
Focus on Shipping Software

- Reward developers to see product to release

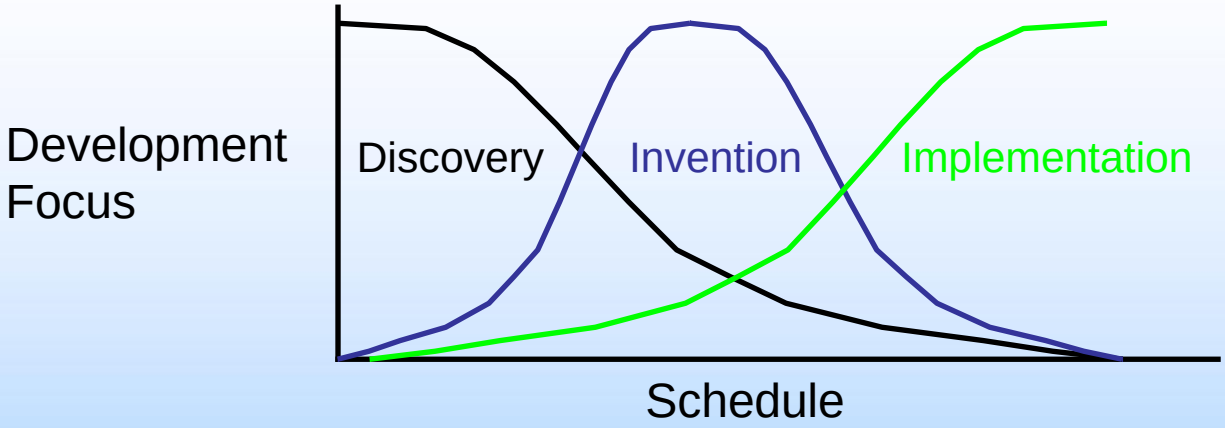
"Microsoft doesn't make money by developing software;
it makes money by shipping software"

- Technical decisions should contribute to satisfying the minimum required functions
- Development is a functional activity that serves practical objectives
- Developers must focus on the bottom line

Successful Project at a Glance

Intellectual Phases

- Project progresses from problem definition to software delivery
  - Discovery
  - Invention
  - Implementation



Successful Project at a Glance

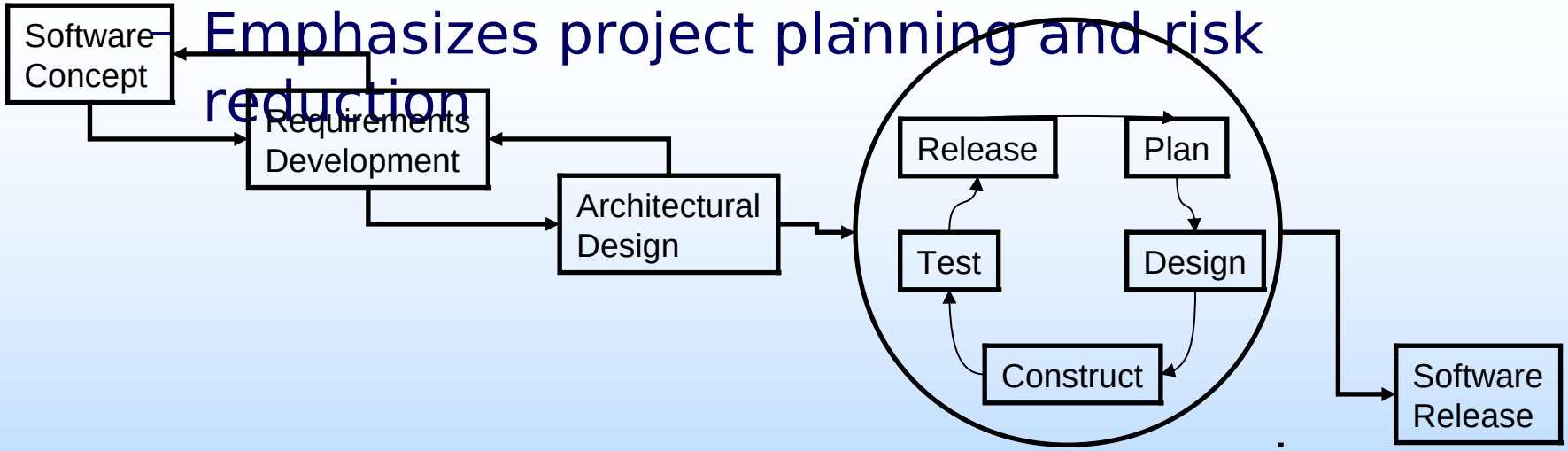
Project Flow

- Status reports of “90% complete” not useful
  - If first 90% took 90% of the time,
  - The last 10% will take another 90% of the time!

Successful Project at a Glance

Staged Delivery

- Deliver in successive stages, not all at once
  - Most important functionality delivered first
  - Easier to track project status
  - Emphasizes project planning and risk reduction



Successful Project at a Glance

Benefits of Staged Delivery

- Critical functionality available earlier
- Risks reduced early
- Less overhead in status reporting
- Reduces possibility of estimation error
  - Software is frequently in a releasable state
- Balances flexibility and efficiency
  - Team only considers changes between stages
  - Guarantees changes will be considered periodically

Successful Project at a Glance

Disadvantages of Staged Delivery

- Overhead
  - Retesting features
  - Version control complexity
  - Releasing multiple times
- But this additional overhead yields
  - Improved status visibility
  - Quality visibility
  - Estimation accuracy
  - Risk reduction

Software Project Survival Guide

Survival Preparations

- Introduction
- The Survival Mindset
- Survival Preparations
- Succeeding by Stages
- Mission Accomplished
- References

Survival Preparations
Outline

- Ch 6: Hitting a Moving Target
- Ch 7: Preliminary Planning
- Ch 8: Requirements Development
- Ch 9: Quality Assurance
- Ch 10: Architecture
- Ch 11: Final Preparations

Hitting a Moving Target
Change Control – Formalized Common Sense

- Track changes in requirements, source code
- Evaluate, control, approve important changes
- Keep stakeholders aware of changes
- 5 Phases
  1. Initial development – No change control
  2. Technical review says what work is done
  3. Completed work sent to change board
  4. Work placed under revision control
  5. Systematically handle further changes

Change Control
Benefits of Automated Revision Control

- Combats “mushy milestones”
- Promotes accountability

  “On successful projects, project members actively seek accountability both for their own work and for other work that affects them.”
- Retrieve any version of any major document
- All documents available publicly
- Those who try it tend to advocate it
- Generates Useful Statistics

Change Control
Common Issues

- How to consider changes
  - Change board considers the benefits and costs
- When to consider changes
  - Change board meets as needed (biweekly to start)
- How to handle small changes
  - Must use common sense or be swamped
- How to handle political issues
  - Let them know they can still get what they want

Preliminary Planning
Basic Idea

- Successful projects begin planning early
  - Define a project vision
  - Identify executive sponsor
  - Set targets for scope
  - How to publicize plans and progress
  - Decide how to manage risks
  - Map out strategies for using personnel effectively
- Capture these in a Software Development Plan

Preliminary Planning
Project Vision

- A study of 75 teams found:
  - Effective teams always have a clear objective
- Effective vision
  - Aids in decision making
  - Has a motivating effect
  - Must be achievable (otherwise it is de-motivating)
  - Makes it easy to know what to include/exclude

Preliminary Planning
Project Vision Examples

- Consider the following vision statements:

  Create the world's best word processor

  Create the world's easiest-to-use word processor

- Is it motivating?
- Does it provide guidance to the dev team?

"Creating wording that excludes at least as much as it includes is the hard part of writing a vision statement, but that wording is essential to the statement's usefulness."

Preliminary Planning
Executive Sponsorship

- Can be a person or a group
- A single, clear decision making authority
  - Else the project gets pulled in different directions like silly putty
- Responsible for committing to a feature set
- Approves the user interface design
- Decides if the software is ready to release

Preliminary Planning
Project Scope Targets

- Software development is process of continual refinement
- Give-take between features, budget, schedule
- Be aware if initial budget and schedule is insufficient for the desired features
- Preliminary schedule is only an aid
  - To identify features that cannot meet target date
  - Remove those features to keep project small

Preliminary Planning
Project Scope Targets

- NASA's Software Engineering Laboratory (SEL)
  - Creates initial project estimate
  - Refines 5 more times over course of the project
  - Estimates have base, upper and lower limits

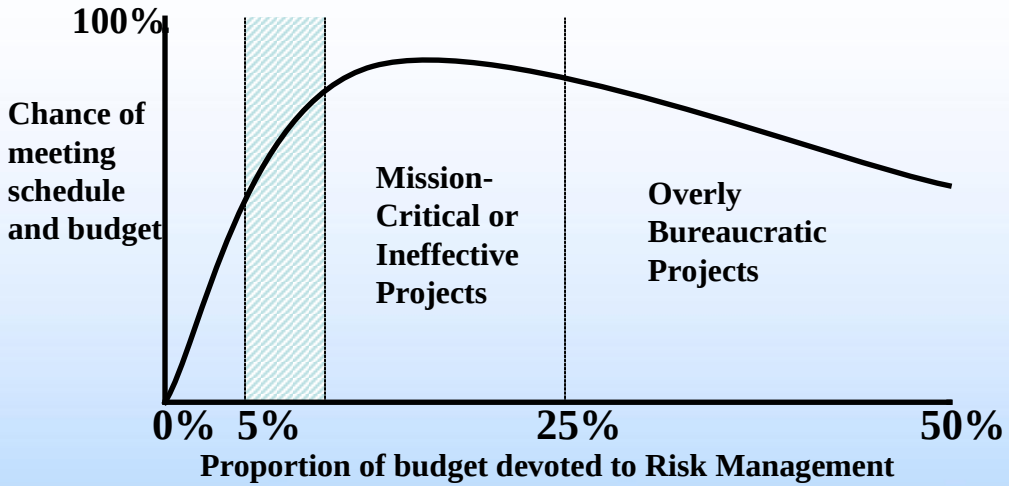
	Upper	Lower Limit
After Requirements Definition	x2.0	x0.5
After Requirements Analysis	x1.75	x0.57
After Preliminary Design	x1.4	x0.71
After Detailed Design	x1.25	x0.80
After Implementation	x1.10	x0.91
After System Testing	x1.05	x0.95

Preliminary Planning
Publicizing Plans and Progress

- Characteristics of unsuccessful projects:
  - Planning is conducted in secret
  - Not on purpose, but effectively so
- Make basic status indicators readily available
  - Good and bad news should be visible
  - List of completed tasks
  - Defects
  - Top 10 Risks List
  - Percent of schedule/resources used
  - PM Status Report to upper management

Preliminary Planning
Risk Management

- This book promotes 5% of project effort to risk management
- Gives most projects a 50 to 75% chance of being on time and within budget



"Successful organizations actively look for ways to trade small amounts of increased overhead for large amounts of risk reduction."

Preliminary Planning
Risk Management

- Three elements to RM commitment
  - Include risk management approach in project plan
  - Budget must include funds for risk resolution
  - Risk impact must be incorporated in project plans
    - Not doing so is like sending your calls to voice mail, but then never checking your messages
- Identify a Risk Officer
  - Should have management's respect
  - If not they become project's designated pessimist

Preliminary Planning
Risk Management – Top 10 Risks List

- Key tool is the Top 10 Risks List
  - Doesn't have to be exactly 10
  - Does have to be maintained
  - Review with project manager regularly
  - Visible to entire team
  - Can include any complaint from anyone
    - Creeping Requirements
    - Released software has low quality
    - Unachievable schedule
    - Friction between developers and customers
    - Unproductive workspace

Personnel Strategies
Personnel Strategies

- People-aware management accountability
  - Do 5 developers quit at the end of a project?
  - Does everyone emerge with improved skills and great morale?
  - Manager is responsible
- Hiring developers: Available vs. good ones

"It is better to wait for a productive programmer to become available than for the first available programmer to become productive."

Personnel Strategies
Productivity Study

- 1993 Research by B. Lakhanpal on 31 teams
  - Which impacts performance more:
    - Individual developer's capabilities
    - Or team cohesion?
- TEAM COHESION had a greater impact
  - Pay attention to how team members work together
  - Do not casually disband good teams after a project

Personnel Strategies
Project Team Roles

- Team Roles
  - Project Manager - Product Manager
  - Architect - User-Interface Designer
  - End-user Liaison - Developers
  - QA/Testers - Tool smith
  - Build Coordinator - Risk Officer
  - End-user documentation specialist

Personnel Strategies
Time Accounting

- Track how personnel spend their time
  - Provides project visibility
  - Better estimations on future projects
- Categories and Activities
  - Management

    Planning, track progress, report progress, manage change, …
  - Architecture

    Create, review, and rework, report defects
  - Integration

    Automate, maintain, test, and distribute the build

Requirements Development
Three Activities

- Convert customer needs into system specs
  1. Gather Requirements
     - Interview potential users
     - Review competing products
     - Build and review prototypes
  2. Specify Requirements
     - Document
     - Storyboards
     - Interactive prototype
  3. Analyze Requirements
     - Break them down to their essential characteristics

Requirements Development
Help Users Define Requirements

- In Steve's own words

"The most difficult part of requirements gathering is not the act of recording what the users want; it is the exploratory, developmental activity of helping users figure out what they want."

Requirements Development
The Prototype

- Build a simple user interface prototype
- Present prototype to key end-users
- Use feedback to revise prototype
- Revise until end-users are excited about it
  - "Highly satisfying" vs. "satisfying requirements"
- Benefits
  - Clarifies requirements
  - Make changes early, when changes are cheap
  - Increases user-involvement

Requirements Development
The Prototype (2)

- Develop style guide from prototype
  - Set standards for the look and feel
  - Few pages long, includes screen shots, fonts, …
  - Review and place it under change control
- Fully extend the prototype
  - Demonstrate every functional area of the software
  - Broad but shallow (show, don't implement)
  - Use as the baseline specification
- No prototype code in the final software

Requirements Development
The Prototype (3)

- Write detailed user document from prototype
  - Near the beginning of the project
  - More understandable to users than technical spec
  - Describes only end-to-end functionality of software
- Create non-user-interface requirements doc
  - Detailed algorithms
  - Interactions with other software and hardware
  - Performance requirements
  - Memory usage
  - Other less visible requirements
  - Reviewed and placed under change control

Quality Assurance
Why Quality Matters

- Low quality software
  - Increases burden on end-user support
  - Increases maintenance costs
- Microsoft solution
  - Charge support costs back to business unit that created it
- Customers remember quality, not delivery date

Quality Assurance
Quality Assurance Plan

- Commitment to QA required for survival
- Elements of QA plan
  - Defect tracking
  - Unit testing
  - Source code tracing
  - Technical reviews
  - Integration testing
  - System testing

Quality Assurance
Defect Tracking

- Defect report includes
  - Description
  - How to reproduce defect
  - Date when defect was detected and fixed
  - Who found it and who fixed it
  - Effort required to fix
  - Phases when defect was created and detected
- Helps project team track project status
- Provides useful info for future projects

Quality Assurance
Technical Reviews

- Walkthroughs, inspections, or code reading
- Focus: detect defects upstream
- Allows jr. developers to associate with sr. developers
- Opportunity for jr. developers to challenge old assumptions

Quality Assurance
System Testing

- Technical reviews improve quality upstream
- System testing improves quality downstream
  - Use independent testers, not the same developers
  - Tests need to cover 100% software's functionality
  - Adequate resources for testing
    - Microsoft: 1 tester / 1 developer
    - NASA: Life critical software, 10 testers / 1 developer
    - Small, in-house business systems: 1 tester / 4 developers
- Testing alone does not assure quality

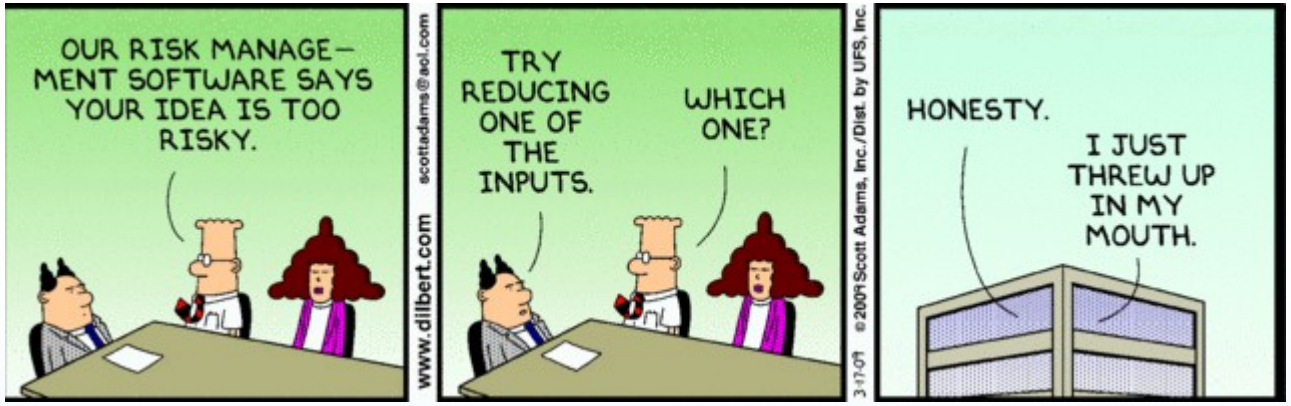
Dilbert Diversion
Break Time

- System testing

- Customer requirements

Dilbert Diversion
Break Time

- Risk Management



- Productivity

Architecture
Goal

"Once you realize that the goal of architecture is to reduce complexity, it becomes clear that the architect must focus as much on what to leave out of the system as on what to put in"

Architecture
Definition

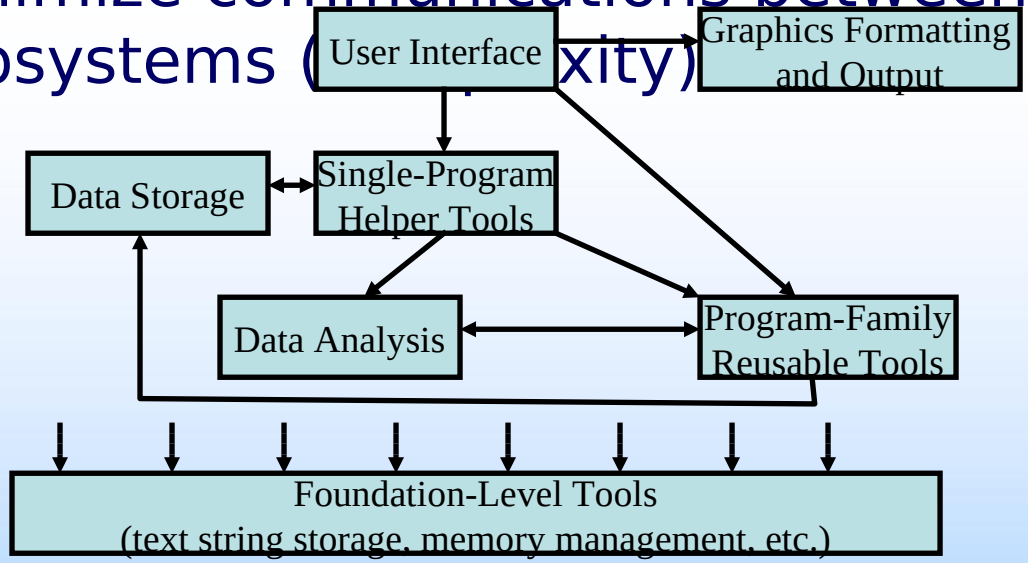
- Make hard-to-change decisions - correctly
- Provides technical structure for project
  - Good: Makes the rest of the project easy
  - Bad: Makes the rest of the project impossible
- Provides blueprint for detailed design
  - How will likely changes be supported
  - Can components from other systems be reused
  - What components will be purchased commercially
  - Design approaches to functional areas
  - How does the architecture address requirements

Architecture
Characteristics

- Plan maintainability, security, testability from start
  - Indicate reasons for selecting solutions
  - Indicate reasons why alternatives were not chosen
- Conceptual Integrity
  - Clearly defined goal
  - The architecture should fit the problem
  - People will say “Of course, obviously, …”
  - Deep Simplicity (natural and simple solution)
  - More complicated architecture is worse, not better

Architecture
Characteristics (2)

- Organization
  - Well defined subsystems, 5 to 9 function clusters
  - Minimize communications between subsystems (



Architecture
Characteristics (3)

- Notation
  - Standard notation like UML for large projects
  - Understood by all project members
- Strategy for handling Change Scenarios
  - Anticipate most likely changes
  - Plan how they will be addressed
- Reuse Analysis and Buy vs. Build Decisions
  - Decision affects costs and schedule
  - Consider reusing code, data, designs, test cases, …

Architecture
Characteristics (4)

- Design decisions that effect implementation
  - External software interfaces
  - User interface
  - Database organization and data storage
  - Key algorithms
  - Memory management
  - Security
  - Concurrency/Threading
  - Portability
  - Programming language

Architecture
Characteristics (5)

- Support for Staged Delivery Plan
  - Identify dependencies between different parts
  - Plan to deliver parts in order that supports stages
- When is it complete?
  - The architecture is never perfect
  - If you strive for the best, you often end up with nothing

Final Preparations

- More detailed plans than in early stages
- When to start?
  - After requirements are baselined
  - Architecture is underway
- Tasks that are part of final preparations
  1. Create project estimates
  2. Write staged delivery plan
  3. Ongoing planning activities

Final Preparations
Project Estimates

- Estimate effort, cost, and schedule
- Rules of thumb to keep in mind
  - Accurate estimate IS possible
  - Accurate estimate takes time
  - Requires quantitative approach (software tool)
  - Use data from projects completed by same group
  - Estimates require refinement

Final Preparations
Project Estimates (2)

- Estimation procedure should:
  - Be written
  - Include time for all normal activities
  - Not assume working overtime
  - Come from estimation software
  - Be based upon data from completed projects
  - Be re-estimated several times during project
  - Be placed under change control
- Watch for estimates developers don't accept

Final Preparations
Staged Delivery

- Most important functionality first
- Reduces time that *seems* to be required
- Design a THEME for each stage
  - Simplifies decision of included features
  - Example:
    - Stage 1 – Database
    - Stage 2 – Billing
    - Stage 3 – Networking
    - Stage 4 – Extended Reporting
    - Stage 5 - Automation

Final Preparations
Ongoing Planning Activities

- Look at plans that were developed earlier
  - Risk Management
    - Update Top 10 Risks a few times by now
    - New risks – cost, resources, technical aspects, …
  - Project Vision
    - Revise if necessary
    - Must provide direction for all project phases
  - Personnel
    - Project Morale, problematic members, organization, additional recruitment, …
  - Software Development Plan

Software Project Survival Guide

Succeeding by Stages

- Introduction
- The Survival Mindset
- Survival Preparations
- Succeeding by Stages
- Mission Accomplished
- References

Succeeding by Stages
Outline

- Ch 12: Beginning of Stage Planning
- Ch 13: Detailed Design
- Ch 14: Construction
- Ch 15: System Testing
- Ch 16: Software Release
- Ch 17: End-of-Stage Wrap-Up

Stage Planning
Beginning-of-Stage Planning

- Why is Stage Planning Needed?
  - Minimize damage caused by shaky estimates
  - Forces developers to have a releasable product
  - Reduces Risk
- Create Individual Stage Plan
  - Tells how to conduct detailed design, coding, …
  - Add it to the Software Development Plan
- Mini Milestones help track progress

Stage Planning
The Stage Plan

- Includes milestones, schedules, and tasks for
  - Risk management
  - Construction
  - Technical reviews
  - Coordination between developers and testers
  - Project tracking (mini-milestones)
  - Updating user documentation
  - Integration and release
  - Updating requirements
  - End-of-stage wrap-up
  - Detailed design
  - Test case creation
  - Defect corrections

Stage Planning
Miniature Milestones

- Labor intensive but vitally important
  - Frequent targets - Daily to Weekly level
  - Binary milestones (Done / Not Done)
  - Keeps team focused better than long-term ones
  - List EVERY task needed to release software
- All projects will have mistakes

"Project success depends on positioning the project team to detect and correct these mistakes quickly and easily"

Stage Planning
Miniature Milestones (2)

- Reduces risk of low-quality software
- Missed Milestones is an early warning
- If frequently missing milestones
  - Recalibrate the schedule
  - Reduce the project scope
  - Reassign parts to other developers
- Let people define their own milestones

Detailed Design
Overview

- Extends work done by architecture
  - Architecture = system level organization
  - Detailed Design = class/routine level organization
- Reuse analysis
- Trace and resolve requirements
- Construction plan with mini-milestones
- Correct defects in the architecture
- Stage 1 design to address potential problems

Detailed Design
Technical Reviews

- 2 or 3 person review team checks for:
  - Correctness (Will it work as intended?)
  - Completeness (Is the design adequate?)
  - Understandability (Is it easily understood?)
  - Meets all requirements
  - Does NOT exceed the requirements
- Facilitates cross-training
- Provides early defect detection

Construction
Keeping it Productive and Fun

- Use a coding standard
  - Prevents a Rube Goldberg contraption
  - Enables maintainability and extensibility
  - Layout of classes, modules, routines, and code
  - Comments
  - Naming variables, functions, and code files
  - Degree of complexity allowed
- Typically enforced during code reviews

Construction
Simplify

- Developers look to optimize and simplify
  - Projects not cancelled from lack of complexity
  - Complexity can make future changes impossible
- Use the project vision to guide decisions

Construction
Software Integration Procedure

- Integration procedure
  1. Developer develops a piece of code
  2. Developer unit tests the code
  3. Developer steps through each line of code including exception and error cases in an interactive debugger
  4. Integrates code with private version of main build
  5. Developer submits code for technical review

Construction
Software Integration Procedure

- Integration procedure (continued)
  6. Developer turns code over to testing for test case prep
  7. Code is reviewed
  8. Developer fixes problems identified during review
  9. Fixes are reviewed
  10. Developer integrates final code with main build
  11. Code is declared “complete”

Construction
Software Integration Procedure

- Requires discipline
- Provides project control benefits
  - "Done" means "Done"
  - Stable foundation for other developer's work
  - Reduces latent quality issues later on

Construction
Daily Build and Smoke Test

- Daily build and smoke test procedure
  1. Merge code changes
  2. Build and test a private release
  3. Execute the smoke test
  4. Check in
  5. Generate the daily build
  6. Run the smoke test
  7. Fix any problems immediately

Construction
Daily Build and Smoke Test

- Every Day – complete rebuild and smoke test
  - Run app and see if something breaks "smokes"
  - Can be automated
  - Ensures/improves quality

Construction
Tracking Progress

- Big job during construction
- Mini-milestones must be tracked
  - Use automated tools such as Microsoft Project
- Similarly with time-accounting
- Provides visibility

Construction
Tracking Progress

- Weekly project tracking update
  - Compare planned vs. actual
    - Defects, milestones, effort, …
  - Top 10 Risks
  - Anonymous feedback
  - Changes proposed and approved
  - Effect on the Schedule or Project Plan
  - Take corrective action if necessary
  - Update software project log
  - Communicate changes to management and users

Construction
Is that all there is to it?

- Construction is THE critical activity
- Everything else supports construction
- Success/failure largely based on groundwork

"If upstream project stages have been conducted effectively, construction will be a time during which a great deal of work takes place <u>uneventfully</u>"

System Testing
Test Philosophy

- System testing at same time as construction
- Verifies requirements are met
- Verifies level of quality
- Test entire system scope
- During each stage
  - Add new test cases to cover new requirements
  - Regression test the previous requirements

System Testing
Testers and Developers Support Each Other

- Testers expose defects to be corrected
- Ensure system quality is high
- Developers correct defects quickly
- PM limits unresolved defects
- Defect tracking identifies problematic routines
  - 80% of system errors found in 20% of routines
  - Review those routines and bring up to standard

Software Release
Drive to a Releasable Level

- Staged delivery relies on bringing the software to a releasable quality level
- Failure to do so results in low quality
- When to Release?
  - Defect counts
  - Effort per Defect (time required to fix)
  - Defect density (defects per 1000 lines of code)
  - Defect Pooling
  - Defect Seeding
  - Defect Modeling

Software Release
Release Checklist

- Avoid simple oversights
- Focus not on testing
- Focus on easily overlooked items
  - Version numbers
  - Removing debugging code
  - Remove seeded defects
  - Smoke/Regression test the Final Build
  - Install on clean system
  - Install over previous version
  - Verify Copyright, license, …

End-of-Stage Wrap-Up

- Opportunity to learn from the experience
- Hold a Change Board Meeting
- Recalibrate Estimates
- Re-estimate Productivity
- Evaluate performance against Project Plan
- Archive project media
- Update the Software Project Log

Software Project Survival Guide

Mission Accomplished

- Introduction
- The Survival Mindset
- Survival Preparations
- Succeeding by Stages
- Mission Accomplished
- References

Mission Accomplished
Project History

- Learn from successes and failures
- Gather data from postmortems or emails
  - What worked
  - What didn't
  - Discuss insights
  - Project Review Questionnaire
- Formalized in the Project History Document
- Don't forget it – Use in future projects

Mission Accomplished
NASA's Success Checklist

- Summary of successful projects
  - DO's
    - Create and follow a Software Development Plan
    - Empower project personnel
    - Minimize the bureaucracy
    - Define the requirements baseline, manage changes to it
    - Take periodic snapshots of project health and progress
    - Re-estimate system size, effort, and schedules periodically
    - Define and manage phase transitions
    - Foster a team spirit
    - Start the Project with a small senior staff

Mission Accomplished
NASA's Success Checklist

- Summary of successful projects
  - DONT's
    - Don't let team members work in an unsystematic way
    - Don't set unreasonable goals
    - Don't implement changes w/o assessing impact, approval
    - Don't gold-plate
    - Don't overstaff, especially early in the project
    - Don't assume schedule slip mid-phase will be made up later
    - Don't relax standards to cut costs or to shorten schedule
    - Don't assume tons of documentation ensures success

Mission Accomplished
Compare with Agile Methods

- Similarities
  - Staged releases
    - Focus on in-phase defect removal
    - Bring software to releasable level regularly
    - Reduced integration risk – small amount being integrated
    - Evidence of progress, adds visibility to customer
    - Increases morale
  - Daily build and smoke test
    - Requires discipline and effort to keep the build healthy
  - Organization Environment
    - Adaptable to many small and medium-sized organizations

Mission Accomplished
Compare with Agile Methods

- Somewhat Similar
  - Agile goals
    - Working software over comprehensive documents
    - Customer collaboration over contract negotiation
    - Individuals and interaction over processes and tools
    - Responding to change over following a plan

Mission Accomplished
Compare with Agile Methods

- Differences
  - Staged releases
    - Agile looks for frequent releases (weekly to monthly)
    - Book doesn't specify but uses 3-month examples
    - Agile does not require all steps in each phase
  - Requirements
    - Agile assumes very dynamic requirements (few up front)
    - Book advises to work hard to firm up requirements early
  - Documentation
    - Agile assumes little or no documentation

Mission Accomplished
Compare with Agile Methods

- Differences
  - Acceptance testing
    - Agile (XP) has on-site customer create acceptance tests
    - Book says testers create tests upon required functionality
  - Team meetings
    - Agile (scrum) promotes daily 15-minute stand-up meeting
    - Book focuses more on visibility of status and progress
  - Code Ownership
    - Agile (XP): any programmer works on any code any time
    - Book: code divided by programmer, strict code reviews

Mission Accomplished
Compare with Agile Methods

- Book balances process with agility
  - Based on project size and complexity
  - Enough process for project survival and success
- In Steve's own words

"I've never been either pro-agile or anti-agile -- I've always been pro-whatever-practices-work-best. In many situations the practices that work best are the practices that today are associated with agile development. And in some circumstances, other older practices still work best."

Epilogue
Bump, bump, bump…

For more than a generation, medium-size projects have been failing for no good reason.

Like Edward Bear, developers, project managers, and customers bump their heads down stairs exactly the same way project after project.

The work required to make a software project succeed is not especially difficult or time-consuming, but it must be executed diligently from the first day of the project to the last.

Software Project Survival Guide

References

- Introduction
- The Survival Mindset
- Survival Preparations
- Succeeding by Stages
- Mission Accomplished
- References

References

- Construx website: http://www.construx.com/survivalguide
- NASA's SEL Recommended Approach to software Development, Rev 3, free online at http://homepages.inf.ed.ac.uk/dts/pm/Papers/nasa-approach.pdf
- Key Practices of the Capability Maturity Model, Version 1.1, free online at http://www.sei.cmu.edu/reports/93tr025.pdf