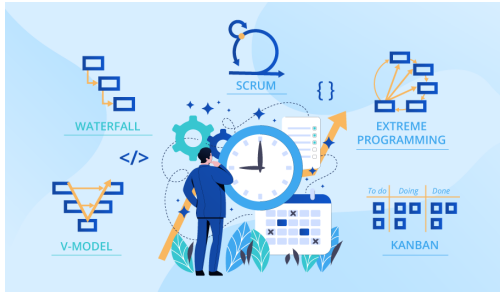


Change Management

SE423: Software Project Management

Outline

- Change Control
- Software Maintenance
- Other Aspects
- Summary

Change Control

Software Change

- Software change is inevitable
  - New requirements emerge when the software is under development or being used
  - The business environment changes
  - Errors must be repaired, Risks mitigated
  - New equipment must be accommodated
  - The performance or reliability may have to be improved
- A key problem for organizations is implementing and managing change to their current projects and legacy systems

Integrated Change Control

- Integrated Change Control is a process concerned with project change requests
- The process is carried out from project inception through completion
- All changes must be carefully controlled to maintain the integrity and consistency of the project plan
  - Change control encompasses review, evaluation, approval/rejection, and managing and coordinating approved changes

Integrated Change Control

- Recognizes that projects will often require changes to the established project plan
- All changes must be carefully controlled to maintain the integrity and consistency of the project plan
- Integrated change control encompasses all aspects of change to the project:
  - Reviewing and approving requested changes
  - Managing the changes when they actually occur
  - Controlling elements of the project management plan (scope, cost, budget, schedule, and quality) in response to changes
  - Controlling changes to requirements, design, code and documentation.

Change or Configuration Control

Configuration Management Plan

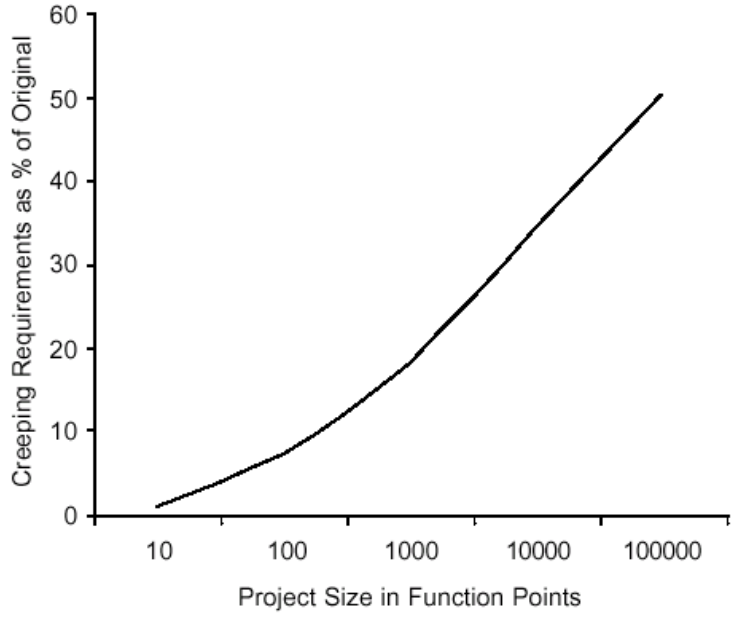
- Change & Version control
  - Items:
    - Code (source for product)
    - Documents: requirements, design, test plans, user guides
    - Plans and databases (MS project, etc.)
    - Scripts for testing
    - Scripts for infrastructure
    - Software development plan and other process documents

Integrated Change Control

- In some projects, the Integrated Change Control process includes a *change control board* (CCB)
- The CCB is a formally chartered group responsible for reviewing, evaluating, approving, delaying, or rejecting changes to the project, and for recording and communicating such decisions
- Note that changes have a potentially greater impact in CPM scheduling:
  - Changes on the critical path have greatest impact while those off the critical path have less
- Changes in CCM scheduling require adjustments to buffers
  - The same responses to change are needed, but the impact to the schedule may be less severe than in CPM

Change Control

- Average project has 25% requirements change
- Overly detailed specs. or prolonged requirements phase are not the answer
- Sources of change
- Change control is a process
- Change Control Board (CCB)
  - Structure, process, triage



Control the Change – I

1. Need for change is recognized
2. Change request is submitted as a “request for change” (RFC) or engineering change order (ECO)
3. Developer or PM team evaluates: impact and desirability
4. Change report is generated
5. Change control authority (CCB) makes a decision to either:
   a) Deny request.
      i. Change request is denied
      ii. User is informed
   b) Proceed

If change is approved:

1. Request is queued for action. ECO (Engineering Change Order) is generated.
2. Individuals assigned to configuration objects.
3. Objects checked out and change made.
4. Change audited.
5. Baseline established.
6. If it is a Software Configuration Item (SCI)
   - Perform SQA and testing activities
7. Check-in the changed objects

Control the Change – II

For Software Configuration Items (SCI)

1. Promote SCI for inclusion in next release
2. Rebuild appropriate version
    a) Include all changes in release
    b) Review/audit the change
    c) Perform Verification and Validation [testing activities]
3. Distribute new version

Control the Change – III

- Use a change management system (COTS) and a change tracking system.
- Ideal if they are integrated: aka Comprehensive Software Change Management
  - SCM (Source code management)
  - Issue/defect tracking software

Agile Perspective (Integrated change control)

- The agile change control process is not controlled by a change review board and the project manager
  - *Product* changes are owned and managed by the customer
  - *Process* changes are owned by the team
  - The project manager facilitates collaborative discussion of changes between the customer and the team

☛ *Integrated Change Control* maps to 'continuous backlog management' in an agile project

Integrated Change Control Comparison

Feature	Plan-driven Methodologies	Adaptive Methodologies
Change Control Approach	Formal and structured approach	Informal and flexible approach
Change Management Plan	Well-defined and documented plan	Limited or no plan
Change Control Board (CCB)	Formal CCB with defined roles and responsibilities	Informal CCB with flexible roles and responsibilities
Change Request Process	Formal process with strict change request procedures	Informal process with flexible change request procedures
Change Impact Analysis	Thorough and detailed analysis of change impact	Limited or no analysis of change impact
Change Implementation	Changes are implemented after approval from CCB	Changes are implemented quickly and frequently
Documented Changes	Changes are well-documented and tracked	Changes may not be documented or tracked
Change Communication	Formal and structured communication plan	Informal and flexible communication plan
Project Management Approach	Emphasizes predictability and control	Emphasizes flexibility and adaptability
Applicability	Best suited for projects with well-defined requirements and a stable environment	Best suited for projects with changing requirements and an uncertain environment

Change During Development

- Sometimes change occurs during development that necessitates changes in scope
  - Approval of CCB (Change Control Board) and
  - Requires extensive planning
  - May require more time/resources (project triangle)
- Plan-driven methodologies may or may not have this built in (i.e. Spiral) or may be specifically built to resist change (i.e. Waterfall)
- Agile Methodologies embrace change
  - Scrum allows for change to the Product Backlog at any time, but manages risk by freezing the current Sprint Backlog
- Stakeholder Communication IS KEY

Software Change Strategies

- Software maintenance
  - Changes are made in response to changed requirements but the fundamental software structure is stable
- Architectural transformation
  - The architecture of the system is modified generally from a centralized architecture to a distributed architecture
- Software re-engineering
  - No new functionality is added to the system but it is restructured and reorganized to facilitate future changes
- These strategies may be applied separately or together

Lehman's Laws

Law	Description
Continuing change	A program must change or become progressively less useful
Increasing complexity	As program changes, its structure becomes more complex Extra resources are required.
Large program evolution	System attributes, (eg, size, time between releases) is ~invariant for each system release.
Organizational stability	A program's rate of development is ~constant.
Conservation of familiarity	The incremental change in each release is ~constant.
Continuing growth	The functionality has to continually increase to maintain user satisfaction.
Declining quality	Quality of systems appear to be declining unless adapted to changing environments
Feedback system	Evolution processes involve feedback systems for product improvement 10

Applicability of Lehman's Laws

- This has not yet been established
- They are generally applicable to large, tailored systems developed by large organizations
- It is not clear how they should be modified for
  - Shrink-wrapped software products
  - Systems that incorporate a significant number of COTS components
  - Small organizations
  - Medium sized systems

Software Maintenance

Software Maintenance

- Modifying a program after it has been put into use
- Maintenance does not normally involve major changes to the system's architecture
- Changes are implemented by modifying existing components and adding new components to the system

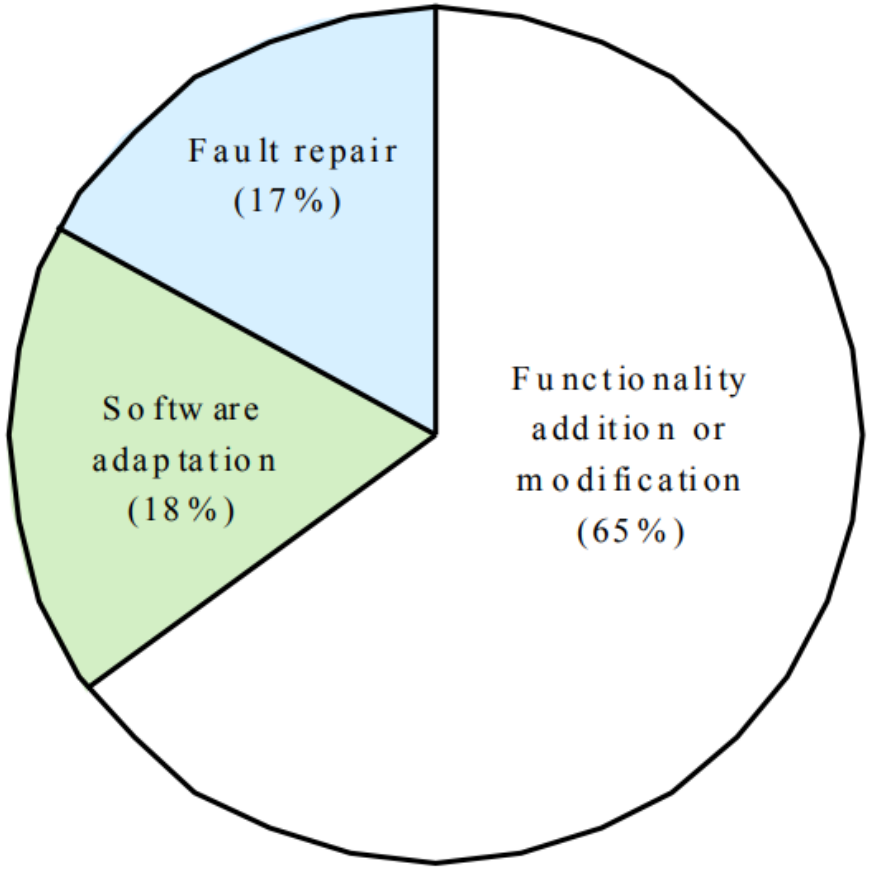
Maintenance is Inventible

- The system requirements are likely to change while the system is being developed because the environment is changing. Therefore a delivered system won't meet its requirements!
- Systems are tightly coupled with their environment. When a system is installed in an environment it changes that environment and therefore changes the system requirements.
- Systems MUST be maintained therefore if they are to remain useful in an environment

Types of Maintenance

- Maintenance to repair software faults
  - Changing a system to correct deficiencies in the way meets its requirements (***Corrective Maintenance***)
- Maintenance to adapt software to a different operating environment
  - Changing a system so that it operates in a different environment (computer, OS, etc.) from its initial implementation (***Adaptive Maintenance***)
- Maintenance to add to or modify the system's functionality
  - Modifying the system to satisfy new requirements (***Perfective Maintenance***)

Distribution of Maintenance Efforts



Maintenance Cost

- Usually greater than development costs (2* to 100* depending on the application)
- Affected by both technical and non-technical factors
- Increases as software is maintained. Maintenance corrupts the software structure so makes further maintenance more difficult.
- Ageing software can have high support costs (e.g. old languages, compilers etc.)

Maintenance Cost Factors

- Team stability
  - Maintenance costs are reduced if the same staff are involved with them for some time
- Contractual responsibility
  - The developers of a system may have no contractual responsibility for maintenance so there is no incentive to design for future change
- Staff skills
  - Maintenance staff are often inexperienced and have limited domain knowledge
- Program age and structure
  - As programs age, their structure is degraded and they become harder to understand and change

Evolutionary Software

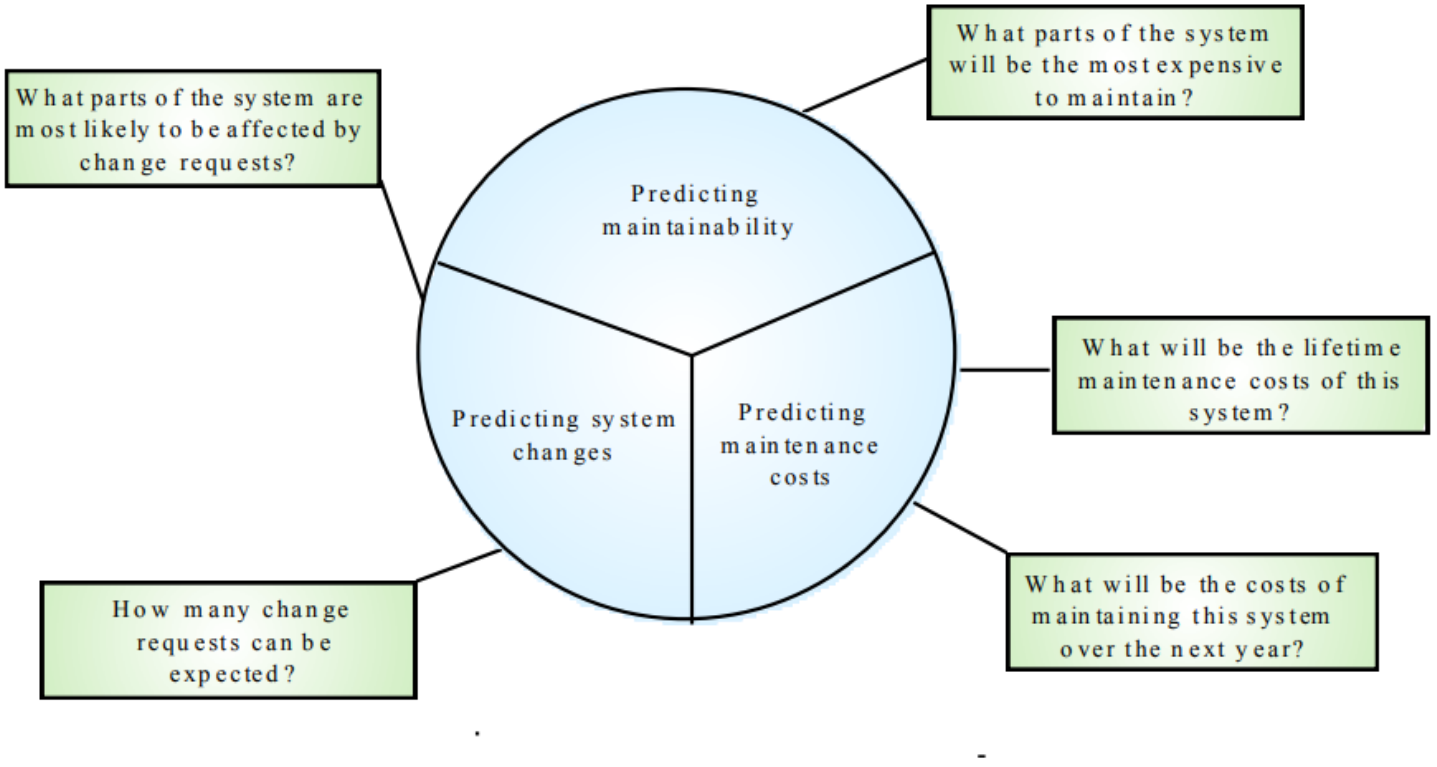
- Rather than think of separate development and maintenance phases, evolutionary software is software that is designed so that it can continuously evolve throughout its lifetime

YES, but how/much?

Maintenance Prediction

- Maintenance prediction is concerned with assessing which parts of the system may cause problems and have high maintenance costs
  - Change acceptance depends on the maintainability of the components affected by the change
  - Implementing changes degrades the system and reduces its maintainability
  - Maintenance costs depend on the number of changes and costs of change depend on maintainability

Maintenance Prediction



Change Prediction

- Predicting the number of changes requires an understanding of the relationships between a system and its environment
- Tightly coupled systems require changes whenever the environment is changed
- Factors influencing this relationship are
  - Number and complexity of system interfaces
  - Number of inherently volatile system requirements
  - The business processes where the system is used

Complexity Metrics

- Predictions of maintainability can be made by assessing the complexity of system components
- Studies have shown that most maintenance effort is spent on a relatively small number of system components
- Complexity depends on
  - Complexity of control structures
  - Complexity of data structures
  - Procedure and module size

Process Metrics

- Process measurements may be used to assess maintainability
  - Number of requests for corrective maintenance
  - Average time required for impact analysis
  - Average time taken to implement a change request
  - Number of outstanding change requests
- If any or all of these is increasing, this may indicate a decline in maintainability

Architecture Evolution

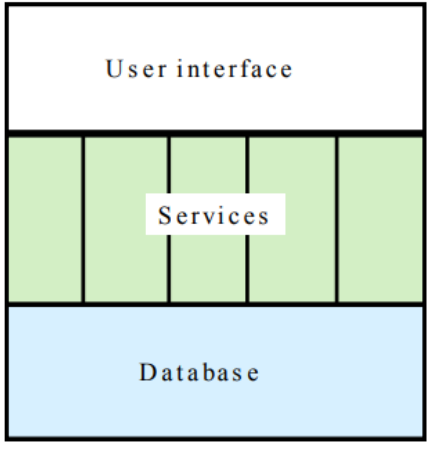
- There is a need to convert many legacy systems from a centralised architecture to a client-server architecture
- Change drivers
  - Hardware costs. Servers are cheaper than mainframes
  - User interface expectations. Users expect graphical user interfaces (CLI->GUI)
  - Distributed access to systems. Users wish to access the system from different, geographically separated, computers

Distribution Factors

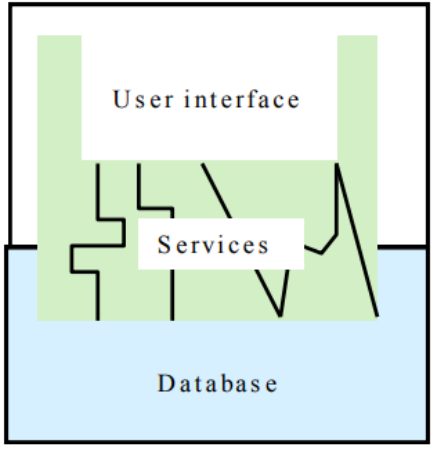
Factor	Description
Business importance	Returns on the investment of distributing a legacy system depend on its importance to the business and how long it will remain important. If distribution provides more efficient support for **stable business processes** then it is more likely to be a cost-effective evolution strategy.
System age	The older the system the more difficult it will be to modify its architecture because previous changes will have degraded the structure of the system.
System structure	The more modular the system, the easier it will be to change the architecture. If the application logic, the data management and the user interface of the system are closely intertwined, it will be difficult to separate functions for migration.
Hardware procurement policies	Application distribution may be necessary if there is company policy to replace expensive mainframe computers with cheaper servers. .

Legacy System Structure

- Ideally, for distribution, there should be a clear separation between the user interface, the system services and the system data management
- In practice, these are usually intermingled in older legacy systems



Ideal model for distribution



Real legacy systems

Key Points

- The costs of software change usually exceed the costs of software development
- Factors influencing maintenance costs include staff stability, the nature of the development contract, skill shortages and degraded system structure
- Architectural evolution is concerned with evolving centralized to distributed architectures
- A distributed user interface can be supported using screen management middleware

Other Aspects

Other Aspects

- Roll-Out
  - Release Check-List
- Training
  - More than just end-users
    - Users, systems ops, maintenance developers, sales
- Documentation
    - Many types: End-user, sales & marketing, operations, design
- Migration
  - Moving users from existing system to your new one
- Maintenance and Support

We'll discuss each of these …

Rollout

- Create a “Release Checklist”
  - Avoid activities falling through the cracks
  - Activities by Group:
    - Engineering, QA, Documentation, Operations
  - Possibly sign-off signatures
- Roll-out: Must have a plan for the process
  - Often on a given day (ex: a Sat.)
  - Must be a very detailed plan

Shipping Details

- Packaging (if commercial product)
- Marketing collateral
- Security mechanisms (if commercial product)
- Licensing
  - Plan
  - Mechanism

Installation

- Scripts
- Uninstall (if not Web-based)
- If you need to install your software (as on PCs):
  - Don’t underestimate:
    - Time this takes to develop
    - Importance of a “first impression”
- Or, if “custom” software you’re reselling
  - Installation at site is often a “mini-project”

Training

- Often more than just end-users
  - Users
  - Sales & Marketing staff
  - System operators
  - Maintenance engineers (possibly)
  - Sales engineers (possibly)
  - (Technical) Support staff

Documentation

- Must be ready by ship-date
- Final user documentation
  - On-line help
- Updates to other
  - Operations documentation
  - Development documentation
  - Sales and marketing material
  - Web site
  - Test reports
  - Release notes

Migration

Migration is the process of moving from the old application/system to the new

- Migration Plan
- Back-out Plan
- Data Conversion

Migration Plan

- Includes
  - Description of environment (computers, DBs, interfaces)
  - Description of existing data needed
  - Description of operational constraints (ex: when can we move to the new system? Weekends only? Last week of month only?)
  - List of affected organizations and contacts
  - Plan of steps to be taken
- Does it require a service interruption?
  - If so, when does this happen? A weekend?
- Training?
- Is there a help-desk?
  - If so, do they have “scripts” or new material?

Migration Strategies

- Communication with customers is crucial
  - Importance of 2-way communication
  - What is happening, when, and why
  - “Why” should remind them of the benefits
  - Not too much detail or too little
  - Where do customers go for more information?
- Minimize intrusiveness
- Find-out about customer’s key dates
  - When does the system absolutely need to be stable?
  - Know about their important deadline dates
  - They must buy-into the approach!

Migration Strategies

- Considerations:
  - Level of business disruption
  - Degree of latitude in “production” date
  - How much internal opposition to system is there?
    - If higher, perhaps a longer ‘adjustment’ period
  - Your comfort level of system quality
    - If questionable, may want to mitigate risk

Migration Strategies

1. Flash-Cut migration
   - ☞ Straight-move from old system to new

   A. Immediate Replacement
      - Fastest approach
      - Still want a back-out plan
      - Requires strong planning and testing

   B. Parallel Operation
      - Mitigates risk
      - Parallel to either existing manual or system process
      - Cut occurs once new system “burned-in”

1. Staged migration
   - Replace one part of existing system at a time

Flash-Cut

- Immediate Replacement
  - Ex: new corporate-wide calendar system
- Requires very careful planning & testing
- Still try to get some users to “try” it first if possible
- Develop a back-out plan

Cutover

- Criteria: What conditions must be met prior?
- Responsibility: Who decides?
- Operations: Who ‘owns’ it once it’s live?
- Rehearsals: Sometimes used.

Back-Out Plan

- Especially important for “conversions”
  - Customers already have expectations and needs as defined by their existing system
  - Must be able to restore customer’s service ASAP
- May mean running both simultaneously “just in case”
- Leave it in place for awhile (more than a day!)
- When to fall-back?
  - Mgmt: sooner, Tech: one-more-fix
  - Set a time limit (ex: 3 hours of start)
- Data recovery and migration back to old system

Data Conversion

- Most systems need this step
- Most PMs forget this
- Impacts both completely new and replacement systems
- The “data” often more valuable than the “system”

Data Conversion Areas

- Data Sources:
  - Where does it come from?
  - Do you need to modify data on the way in?
  - Is it accurate?
- Process Controls:
  - Does it happen all at once?
  - How do you guarantee it’s been done correctly?
- Completion:
  - How do you handle any ‘exceptions’?
  - Do you make backups? Can you restart?

Parallel Operation

- Multiple variations of this method
- An “adoption” period
  - See telephone industry with new area codes
  - Both work for a period of time
- Strategies
  - Avoid flash-cuts if possible
    - Start with test subjects

Concluding Software Projects

- Seems simple, often isn't
- Potential Issues
  - Last-minute change requests
    - "One more feature"
  - Disputes of fulfillment of all requirements
    - Often "interpretation" issues
  - Keeping the team motivated
  - Difficult transition to maintenance

Maintenance Phase

- Need a support plan and a maintenance plan [should be part of project plan]
- The “No respect” phase
- Less “glamorous”
  - Lack of enthusiasm
- Pressure to make fixes quickly
  - For “production” problems
- Software can become “hacked” “patchwork” over time
- Finding a support & test platform can be difficult
  - Often the forgotten child until fixes are needed

Maintenance Phase

- Compare to hardware maintenance
  - Not to keep state same; but changes to state
  - Fixes and enhancements
- Configuration control is very important
  - Fixing the “right” version; tracking branches
- Project management not always carried over
- Smaller team
  - Often not a ‘dedicated team’
  - Drawn from developers with other main tasks

Maintenance Phase

- Contracts, remember those?
  - Always consider the maintenance phase here
  - Often via a “labor hours” contract
    - Time & materials in a “direct” scenario
  - Otherwise via “maintenance contract”
    - Percentage of software license fee
    - Ex: 20% of original cost per year
- Corp. budget if internal/IS projects
  - Often annual/monthly “maintenance” allocations

Project in Trouble

"What if the project cannot meet the schedule?"

- Level with the sponsor
- Move some features/requirements to a second phase
- Use Resource Leveling Techniques
  - Fast tracking – two activities in parallel
  - Activity shifting – Move start/end dates forward or backward
  - Activity splitting – Break an activity into two or more pieces
  - Activity stretching – Use less of a given resource continuously
  - Resource substitution – Assign a different resource
  - Allocating overtime – Work resources longer
- Re-evaluate tasks: effort and need

Project Recovery

How to save a “drowning project”

- 3 Approaches
  - Cut the size of the software
  - Increase process productivity
  - Slip the schedule, proceed with damage control
- Opportunity for decisive leadership action
- Not a time to ‘just cut corners’
  - Be realistic (not foolish)
- Timing: politically important
  - Not too early, not “too” late

Project Recovery

- **First Steps**
  - Assess situation
    - Is there a hard deadline, what's negotiable, etc.
  - Don't do what's been done already
  - Ask team what needs to be done
- **People Steps**
  - Morale; focus; re-assign
    - Restore morale
      - Sacrifice a sacred cow [See note]
        - Dress code, off-site, catered meals, etc.
      - Cleanup personnel problems
    - Focus people's time
      - Remove non-essential work
    - Reassign tasks and responsibilities

Project Recovery

- **Process Steps**
  - Fix classic mistakes
    - Inadequate design, shortchanged activities, etc.?
  - Create “Miniature Milestones”
    - Small (in day(s)), binary, exhaustive
    - Boosts morale: getting things done!
  - Track progress meticulously
  - Recalibrate after a short time
  - Manage risk painstakingly

Project Recovery

- **Product Steps**
  - Stabilize the requirements
  - Raise the bar on change requests
  - Trim the feature set (see feature set control)
    - Determine priorities, cut the low ones
  - “Take out the garbage”
    - Find error-prone modules; re-design
  - Get to a known, stable state & build from there

Feature Set Control

- Minimal Specification
- Requirements Scrubbing
  - unnecessary or overly complex requirements to be removed.
- Versioned Development
- Effective Change Control
- Feature Cuts

Summary

- The software change management process is the process of managing changes to the project scope, schedule, or budget.
- The change management process starts with change identification, where a change request is submitted and recorded.
- The change is then evaluated to determine its impact on the project, including the scope, schedule, budget, and quality.
- If the change is approved, it is implemented and integrated into the project plan.

Summary

- Finally, the change is monitored and controlled to ensure that it does not have any negative impacts on the project.
- Change management techniques include developing a change management plan, establishing a change control board, and communicating changes to stakeholders.
- Change management plans should include the change management process, roles and responsibilities, communication and reporting, and change control procedures.

Summary

- A change control board is responsible for evaluating and approving changes, and ensuring that changes are managed effectively.
- Communication is critical to change management, and project managers should communicate changes clearly and effectively to stakeholders.
- Effective change management can help ensure that changes are managed in a controlled and structured manner, minimizing the risk of negative impacts on the project.