CSC 402, Week 2

Team Organization
Project Planning
Methodology and Process Review

I. **Weekly Overview**

A. ***Monday***:

1. Final review of interview script and schedule.

2. Get started with Milestone 2 tasks, in particular team organization.

Weekly Overview, cont’d

B. ***Wednesday:***

1. *Very brief* reports on Tues/Wed interviews.
2. Overview of 308/402 process and SOPs.
3. Lab meetings with teams, in particular:
   a. Usability 3:10 - 3:30 with Kurfess
   b. Other teams meet, check out repo

Weekly Overview, cont’d

The magic checkout incantation:

```
svn checkout svn+ssh:
  //user@scheduler.csc.calpoly.edu/
    repo scheduler
```

Weekly Overview, cont’d

C. ***Friday:***

1. Round-robin job description presentations.
2. Reports on client xinterviews so far.

Weekly Overview, cont’d

3. Team reports, per Milestone 2 task lists:
    a. Requirements plans
    b. Prototyping plans & demo
    c. Intial marketing plans
    d. Initial usability plans
    e. Initial artistic vision plan

II. Friday Announcements:

A. CSC 402,405,406 Course wiki is up

1. `wiki.csc.calpoly.edu/40256`
2. Use CSL credentials to login.

Friday Announcements, cont'd

B. Admin and content edits have been made to project; to continue over weekend.

C. Scheduling for next week's team work:

1. Monday managers meeting
2. Monday aesthetics meeting
3. Monday feature prioritization discussion

III. **Overview of 308/402 Methodology**

A. Process -- traditional but iterative

B. Artifacts -- dual of process steps

C. Refer to 308 lecture notes weeks 1-4, and accompanying examples

Quick Review of Methodology, cont'd

D. **Highlights of Proposed Standard Operating Procedure**

1. A scheduler project instance has *all* project artifacts: development, wiki, and public web-site.

Quick Review of Methodology, cont’d

a. Development artifacts follow 308 organization. Physcially, the development tree is rooted in a directory named "`scheduler`".

Quick Review of Methodology, cont'd

b. Wiki has ideas, rough plans, other random thoughts; some may be promoted to other development artifacts, as they mature.

- Artifactwise, it's an `adminstration` subdirectory.
- Physically, it's wherever the wiki management software needs it.

Quick Review of Methodology, cont’d

c. Public website is a *selected view* of the project, directed at non-technical clients.

- Artifactwise, it’s a `documentation` sub-directory.
- Physically, it’s linked into `scheduler.csc.calpoly.edu/var/www/`.

Quick Review of Methodology, cont'd

d. The releases are three levels -- alpha, beta, and production.

- Artifactwise, they each contain a checked-out project.
- Physically, they're at `sched-uler.csc.calpoly.edu/release/....`

Quick Review of Methodology, cont'd

e. SVN repository has version-controlled artifacts. Physically, it's at `scheduler.csc.calpoly.edu/repo`.

Quick Review of Methodology, cont’d

2. Individual artifact ownership in the repo.

Quick Review of Methodology, cont’d

a. For artifacts you own, put preliminary or unfinished ideas as draft elements of specific artifacts; it’s OK to SVN commit preliminary ideas, as long as they are in non-operational form, and we’re well in advance of a release date.

Quick Review of Methodology, cont’d

b. For artifacts you don’t own, put preliminary or unfinished ideas in appropriate section of the project wiki; consult with artifact owner(s) about possible inclusion in the repository.

IV. Some of 308/402 Process Details

A. I propose using 308 process and SOPs in 402.

B. We’ll discuss alternatives today.

C. A substantial excerpt of 308 Lecture Notes Weeks 1 and 2 follows.

   1. We’ll cover selected parts in 402 lecture.

   2. You can read over the remainder.

V. The people involved with software.

A. The following are software "stakeholders":

1. ***end users***
2. ***customers***
3. ***domain experts***
4. ***analysts***

Software people, cont’d

5. ***implementors***
6. ***testers***
7. ***managers***
8. ***visionaries***
9. ***maintainers and operators***
10. ***other interested parties***

Software people, cont’d

B. Firt two groups knowns as *"clients"*.

C. First four groups work together.

D. Implementation may not participate in require-ments spec development.

Software people, cont'd

E. In 308, you are primarily analysts, secondarily domain experts and end users.

F. Program design and imple'n happens in 309.

For Your In-Class Viewing Pleasure[†]

- print paper from PDF
- view PDF electronically
- view HTML electronically
- watch main screen and ponder

[†] And in compliance with CPPRA 09.

VI. The software development process.

A. Proper engineering uses an orderly process.

B. Figure 1 depicts major steps.

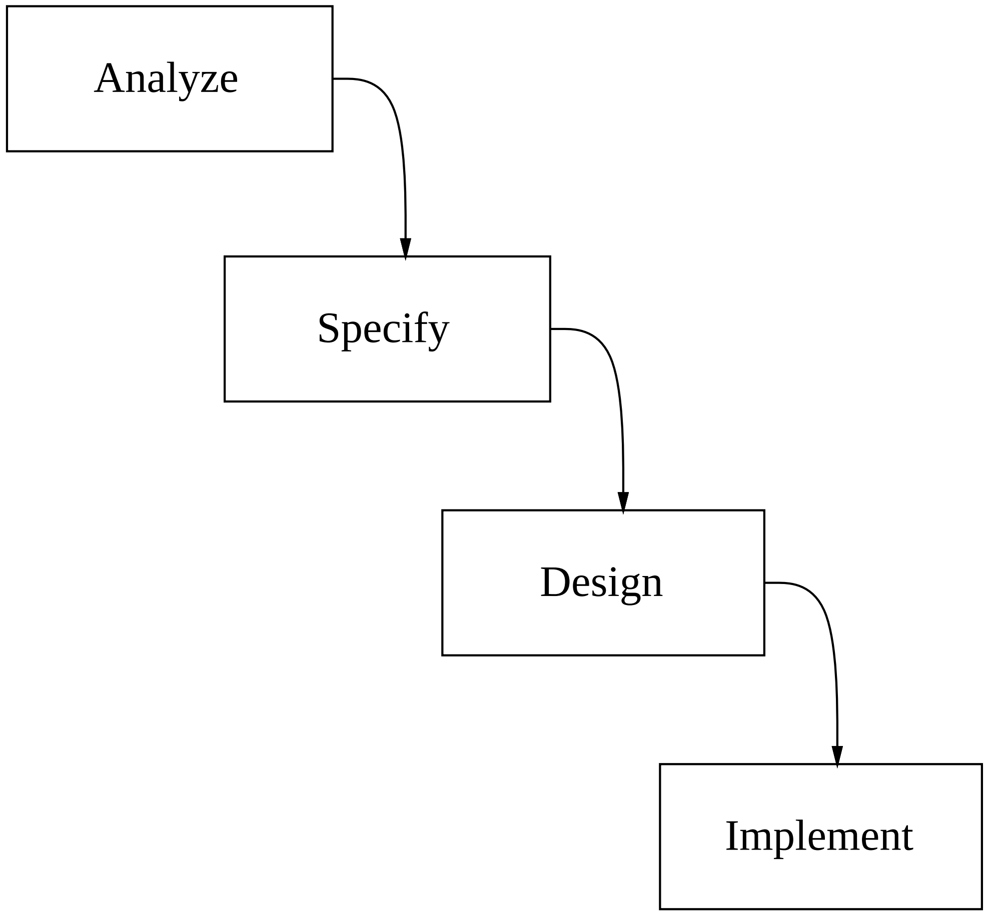


Figure 1: Major phases of SE process.

Process, cont'd

C. The **Analyze** step addresses requirements.

1. Acquire and organize functional requirements of human users.

2. Involves considerable human-to-human communication.

Process, cont’d

D. The **Specify** step involves formal modeling of requirements.

   1. Model can be mechanically analyzed.

   2. Checked for completeness and consistency.

Process, cont’d

E. The **Design** step involves organizing major software components.

   1. Initial design derived from spec model.

   2. Refined into software architecture.

Process, cont’d

F. The **Implement** step fills in operational details.

1. Data structure details are determined.
2. Code for methods is implemented.

Process, cont’d

G. Noteworthy process considerations.

1. Ideally, each step is completed before the next.

   a. Figure 1 seen as a "waterfall chart".

   b. Information only flows down.

Process, cont'd

2. In practice, ideal waterfall is rarely possible.

   a. Water sometimes flows up.

   b. Allows feed-back from lower to higher steps.

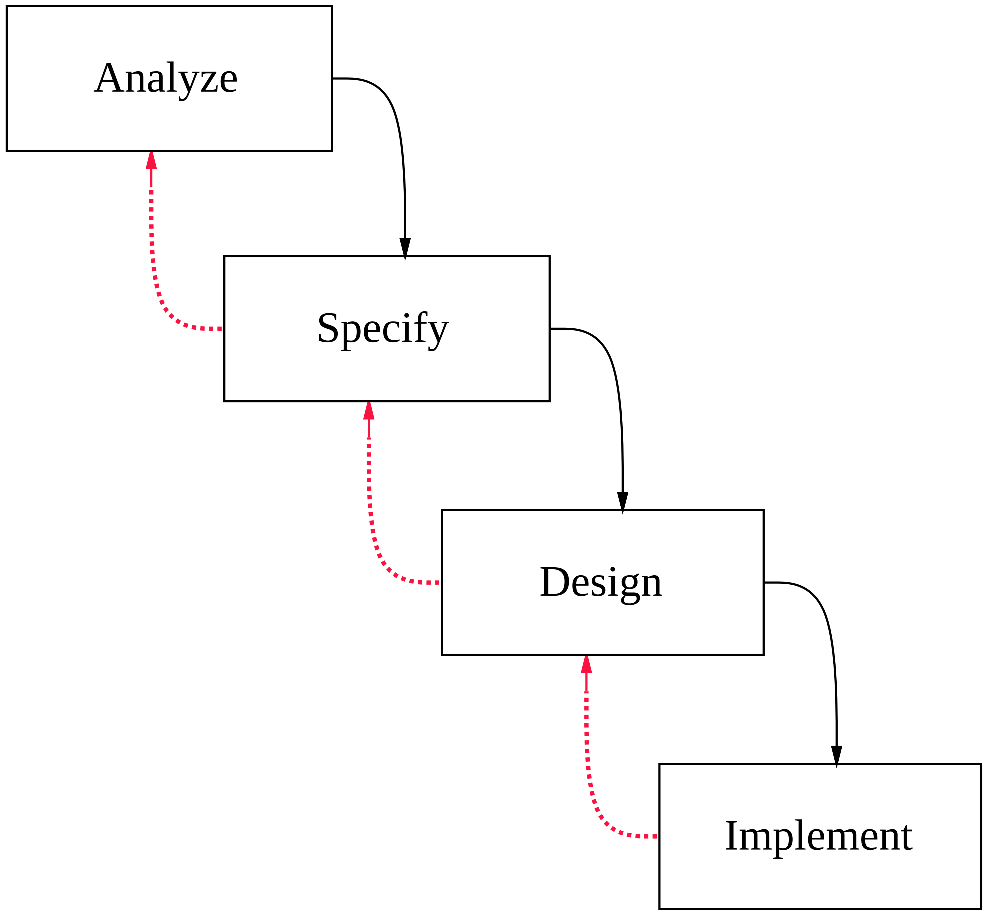


Figure 1: Updated SE process.

Process, cont’d

3. In the 308/309 process:

   a. Much feedback between Analyze and Specify

   b. Much feedback between Design and Impl’t.

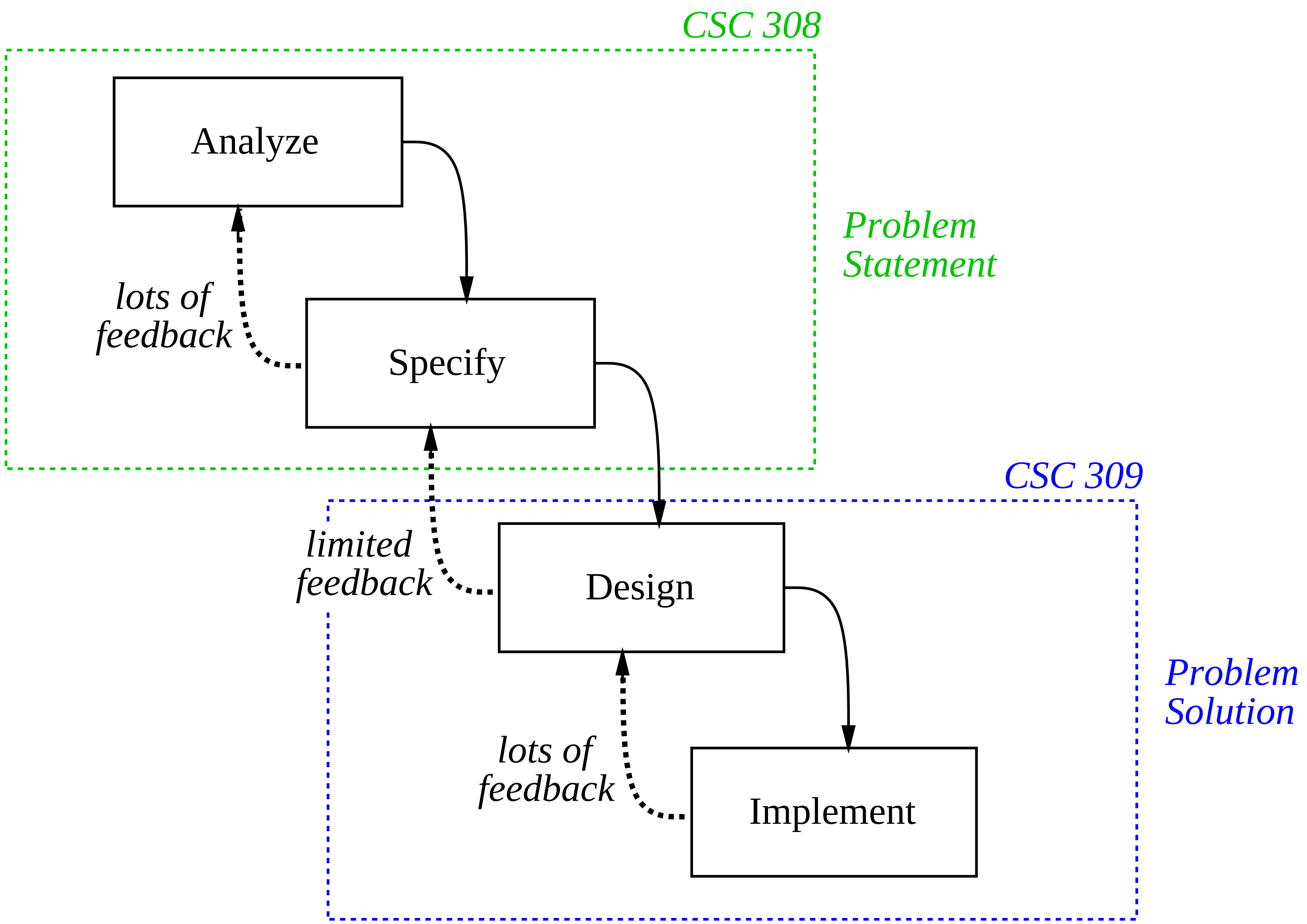
   c. Feedback from Design back up is limited.

Process, cont'd

H. Viewing process as problem solving.

1. Requirements analysis and specification are "problem statement".
2. Design and implementation are "problem solution".
3. Requirements change after design is like changing problem during solution.

Process as problem solving, cont’d



VII. **Pervasive steps of the software process.**

A. Figure 1 shows ordered process steps.

B. Even with feedback, overall order is analyze, specify, design, implement.

C. There are other steps that happen continuously, or "pervasively", throughout process.

Pervasive steps, cont’d

D. The pervasive steps of the process are

1. Manage
2. Configure
3. Test
4. Document
5. Reuse

Pervasive steps, cont’d

E. The **Manage** step entails management of people involved in the process.

1. Project meetings are scheduled at regular intervals.

2. Project supervisors oversee and evaluate the work of their subordinates.

Pervasive steps, cont'd

F. The **Configure** step entails organization and management of software artifacts.

1. Supported by version control tools.

2. The tools manage a software repository.

Pervasive steps, cont'd

G. The **Test** step ensures artifacts meet measurable standards.

1. Testing requirements involves careful human inspection.

2. Testing spec and design involves formal analysis.

3. Testing implementation involves formal functional testing.

Pervasive steps, cont’d

H. The **Document** step produces documents suitable for everyone involved.

1. Requirements spec document.
2. Maintenance documentation.
3. Project reports.
4. End user manuals and tutorials.

Pervasive steps, cont’d

I. The **Reuse** step evaluates existing artifacts to determine if they can be reused.

   1. Reuse from libraries is normal.

   2. Reuse of other artifacts involves refining and adapting.

Pervasive steps, cont'd

J. Important characteristics of pervasive steps.

   1. Carried out during each ordered step.

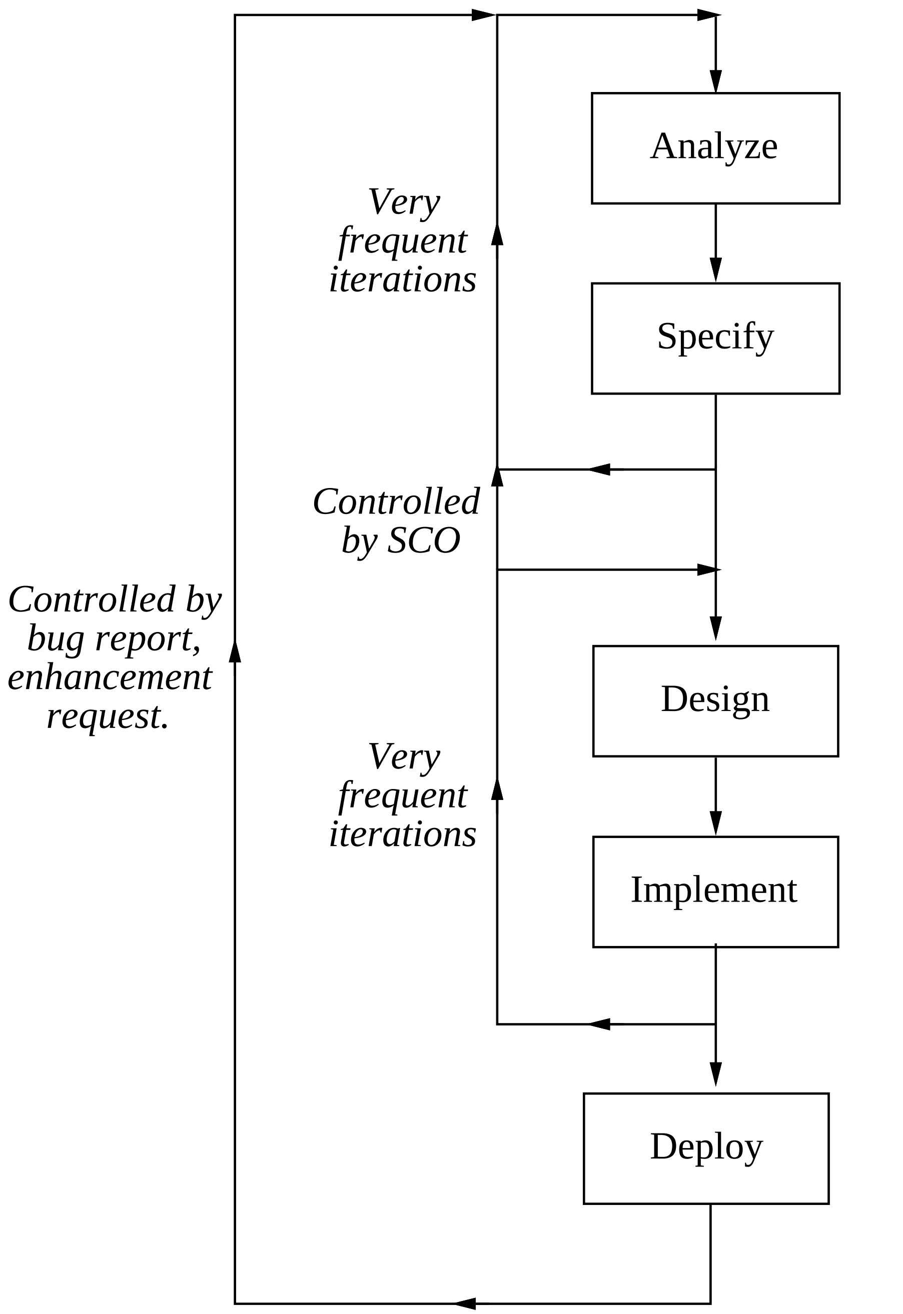
   2. Performed at regularly scheduled intervals.

VIII. **Traditional process versus agile processes.**

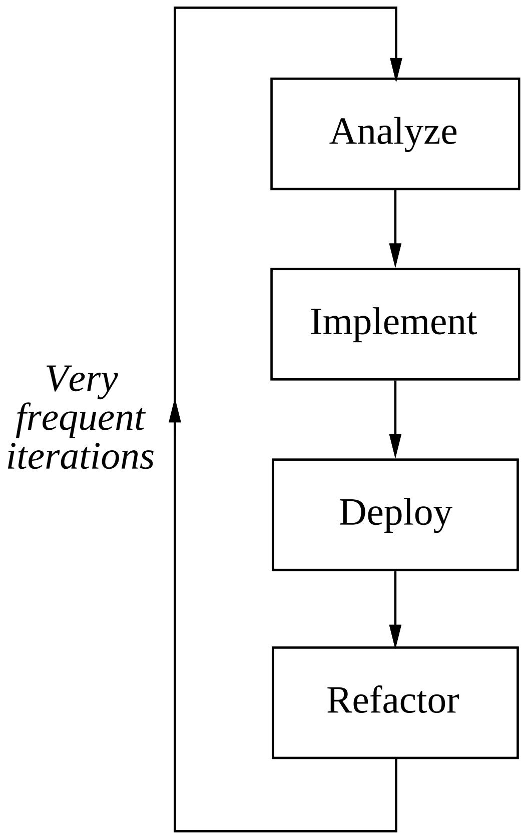
A. 308/309 process considered *traditional*.

B. Particularly the production of a substantial requirements document.

C. More incremental is *agile development*.



a. Traditional process



b. Agile process

Traditional versus agile, cont’d

1. Customers and implementors work very closely together.

2. Traditional steps of specification, design replaced by "refactoring".

Traditional versus agile, cont’d

D. Agile development, extreme programming are relatively new.

1. People have reported success.

2. Few solid studies.

3. Serious question about scale up.

4. At present, some controversy.

IX. **Requirements analysis and specification**

A. Precisely specify need.

B. In a requirements specification document.

C. Informal sections of document understandable to everyone.

D. Formal sections precise enough for contractual instrument.

X. **Importance of careful analysis.**

A. We must have a precise understanding of exactly what user needs are.

B. A seemingly obvious idea.

C. Lure of technology may lead to insufficient time spent on requirements.

Importance of analysis, cont’d

D. Organizations learn that hastily-acquired systems can cause problems.

E. Companies find insubstantial markets for their software products.

F. Nearly universal agreement that thorough requirements analysis is important.

XI. **Patience is required.**

A. Things may seem obvious.

B. Many think they have a clear idea.

C. Everyone may not have *same* idea.

D. Precise analysis helps everyone agree.

XII. Major phases of requirements specification

A. End-user scenarios.

1. Language used is English and pictures.
2. Primary audience is customers, end users.
3. Much user consultation required.

Major phases, cont’d

B. Formal model specification.

1. Formal spec language is used.
2. Primary audience is system design/implementation team.
3. Final version is a *very* formal.

XIII. **Details of user consultations**

A. Critically important to involve end-users in requirements process.

B. Success is far more likely.

C. Many serious failures have resulted when end users are neglected.

XIV. **Activities of user consultation**

A. User interviews.

B. User interface scenarios.

C. User questionnaires or surveys.

D. Visits to other similar installations.

E. Rapid system prototypes.

XV. **Interview techniques**

A. Minimize computer jargon.

B. Specialize questions to each user.

C. Use common sense -- be prepared, polite, succinct, non-threatening, diplomatic, empathetic.

XVI. User interface scenarios

A. Provide users with a concrete view.

B. Premise: "Suppose the system existed already, what would it look like?"

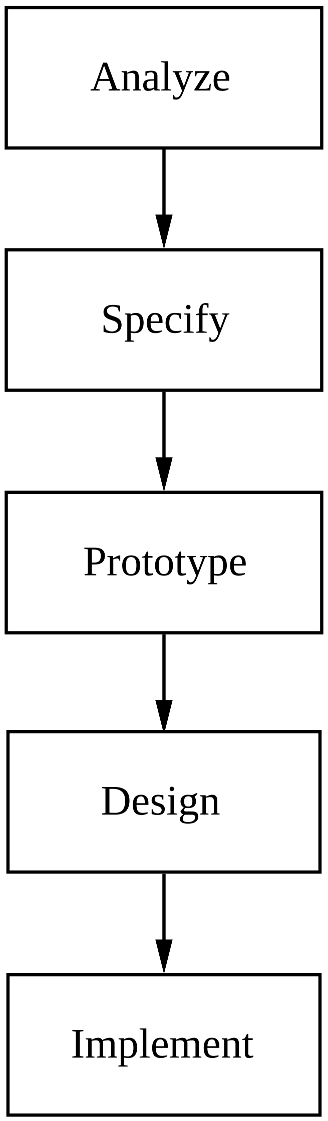
1. Define precisely what user sees.
2. Screens, commands, data formats, and all other user-visible aspects of operation.

XVII. **Rapid system prototyping**

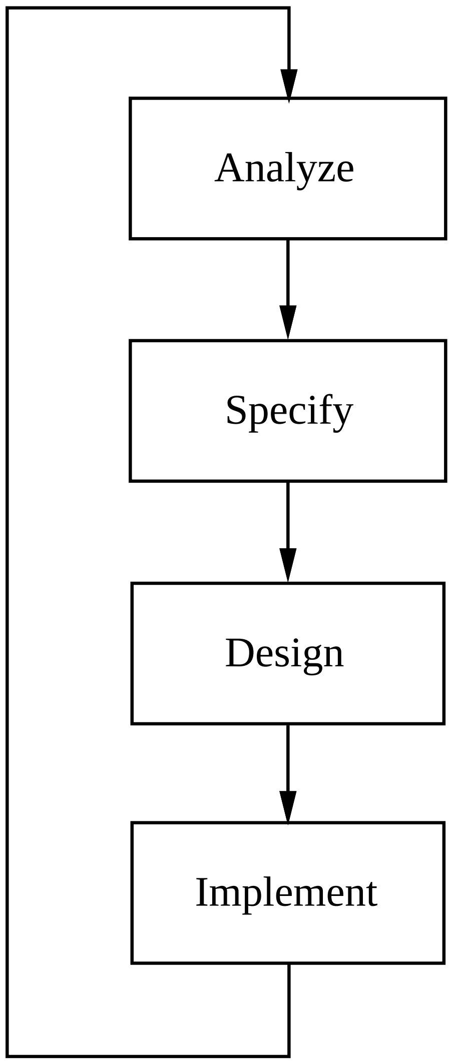
A. Can help capture user requirements.

B. Version with reduced functionality.

C. Figure 2 shows two views or prototyping.



a. As explicit process step



b. As multiple passes

Prototyping, cont'd

D. In 308/309, we'll do both styles

- We'll do a bit of GUI prototyping in 308, as in Figure 2b.
- Overall, the 309 product can be considered an operational prototype, as in Figure 2a.

XVIII. **Establishing genuine user needs**

A. Quite critical.

B. Plenty of software has been built without sufficiently demonstrated need.

C. Forthright analyst should be prepared to say to customers "You don’t need new software"

D. Marketing analysts must be prepared to recognize insubstantial market.

XIX. **Other important aspects**

A. Identification of personnel.

B. Overview of current and proposed operations.

C. Analysis of relevant existing systems.

D. Impact analysis.

Milestone 1 Writeup

- Due friday week 1, check in by 7PM
- Tasks:
  a. Team duties
  b. Brainstorming
  c. Tools search
  d. Questions for week 2,3 customer interviews
  e. Rough draft of Section 1

XX. **Examples of requirements specification**

A. Concrete example similar in size and scope to your 308 projects.

B. Example presented in phases corresponding to milestones.

C. First example covers Milestones 1 and 2.

D. We’ll go over in detail.

Section 1: Introduction

- Initial paragraphs are executive summary.
- Use present tense, third person, active voice.
- Use Calendar Tool example as overall guide.

Section 1.1: Problem Statement

- Succinct presentation of problem(s) to be solved.
- You may (or may not) include the problem of providing a pedagogical example.

Section 1.2: System Personnel

- Description of all people involved.
- For M1, focus on end user categories.
- E.g., for Calendar Tool categories are:
  - registered users
  - group leaders
  - master admins
  - unregistered users

Section 1.3: Operational Setting

- Environment in which tool is used.
- Describe before and after proposed system is installed.
- Consider if proposed system must interface with existing systems.

Section 1.4: Impact Analysis

- Positive, negative impacts in proposed setting.
- E.g., for Calendar Tool:
  - *Positive:* increased convenience and efficiency.
  - *Negative:* decreased privacy, potential disruption of business.

Section 1.5: Related Systems

- Other software with similar functionality.
- Consider:
    - What is good about them.
    - What is bad.
    - What is missing.

Milestone 2 Example

- Very rough draft of requirements.
- Sections 1 and 2 of requirements doc.
- Calendar project is similar to yours.
- Editorial notes provide explanation.
- For M2, focus on content primarily.

Section 2: Functional Requirements

- Definition of all functions and data.
- In scenarios depicting end-user interactions.
- Scenarios are in tutorial style.
  - Tell interesting and engaging story.
  - Give step-by-step presentation.
  - Eventually cover all functionality.

Section 2.1: User-Interface Overview

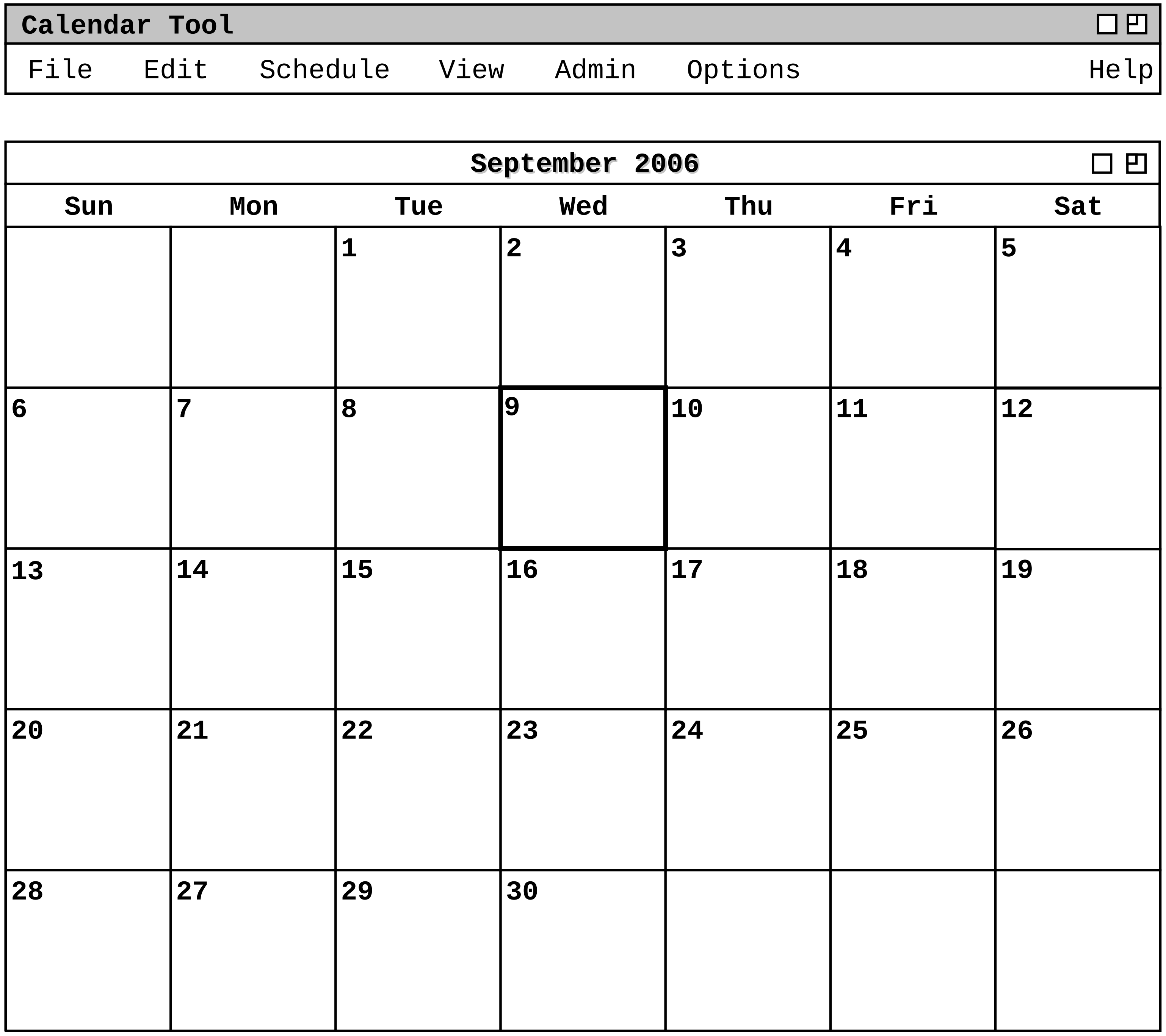
- Standard section for all projects.
- Objective is to present functional hierarchy of tool operations.
- We’ll use pulldown menus as (initial) concrete representation.

UI Overview, cont'd

- Note use of *very simple* GUI.
- More on GUI conventions in next Friday lab.
- ***IMPORTANT:*** Do not get bogged down in low-level GUI details in early stages of requirements.

UI Overview, cont’d

- Start with "When the user initially invokes ..."
- Figure 1 shows initial default screen.
- E.g., here’s Figure 1 for Calendar example:



September 2006

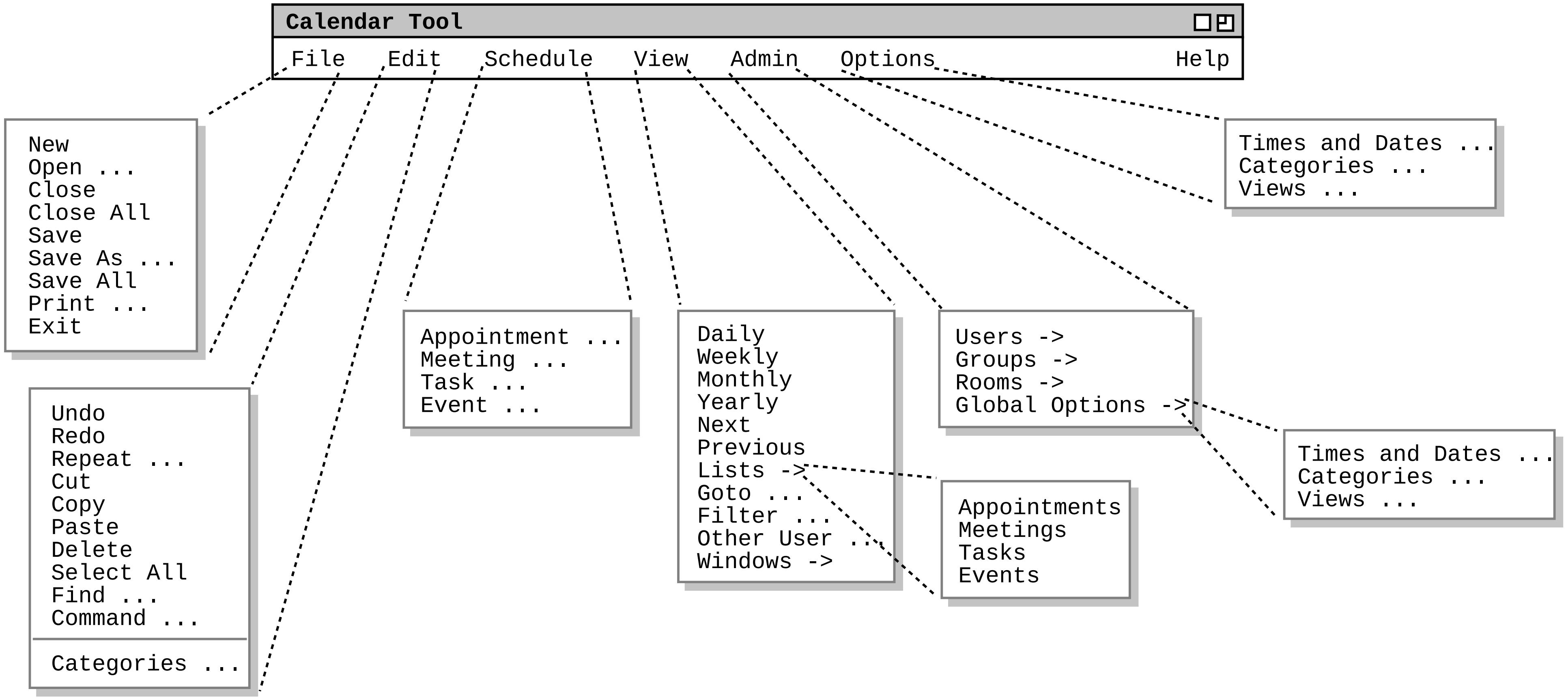
Sun	Mon	Tue	Wed	Thu	Fri	Sat
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
28	27	29	30			

UI Overview, cont’d

- This is how system starts "out of the shrink wrap" for typical user.
- Prose narrative following screen shot explains its contents.

UI Overview, cont’d

- Figure 2 shows expansion of command menus.
- Concrete representation of pulldown menu is convenient standard format.
- Conceptually, we are presenting a
  functional command hierarchy.
- E.g., here’s Figure 2 for Calendar example:


Calendar Tool
File
Edit
Schedule
View
Admin
Options
Help
New
Open ...
Close
Close All
Save
Save As ...
Save All
Print ...
Exit
Undo
Redo
Repeat ...
Cut
Copy
Paste
Delete
Select All
Find ...
Command ...
Categories ...
Appointment ...
Meeting ...
Task ...
Event ...
Daily
Weekly
Monthly
Yearly
Next
Previous
Lists ->
Goto ...
Filter ...
Other User ...
Windows ->
Appointments
Meetings
Tasks
Events
Users ->
Groups ->
Rooms ->
Global Options ->
Times and Dates ...
Categories ...
Views ...
Times and Dates ...
Categories ...
Views ...

UI Overview, cont’d

- A pulldown menu is not the only way to represent a functional command hierarchy.
- It’s a widely-recognized UI standard, at present.
- Next slide shows equivalent functional hierarchy in plain text form.
- Plain text form is acceptable for Milestone 2.

File:
- New
- Open
- Close
- Close All
- Save
- Save As
- Save All
- Print
- Exit

Edit:
- Undo
- Redo
- Repeat
- Cut
- Copy
- Paste
- Delete
- Select All
- Find
- Command
- Categories

Schedule:
- Appointment
- Meeting
- Task
- Event

View:
- Daily
- Weekly
- Monthly
- Yearly
- Next
- Previous
- Lists:
  - Appointments
  - Meetings
  - Tasks
  - Events
- Goto
- Filter
- Other User
- Windows

Admin
- Users
- Groups
- Rooms
- Global Options:
  - Times & Dates
  - Categories
  - Views

Options:
- Times & Dates
- Categories
- Views

Sections 2.2 and Beyond

- These sections differ for each project.
- For Milestone 2 they're rough and preliminary.
    - Calendar example is top-down in style.
    - I.e., a detailed outline has been completed.

2.2 and Beyond, cont'd

- Organizational guidelines:
  - Generally, organize sections per the functional hierarchy.
  - Refine organization with stylistic guidelines, to make document more readable.

2.2 and Beyond, cont’d

- Stylistic guidelines include:
  - Start with scenario showing common activity as a "reader warm up".
  - Simple scenarios first, details later.
  - Separate scenarios for different user groups.

2.2 and Beyond, cont’d

- Leave mundane details until later, e.g., File, Edit, fine details of data entry.
- Leave details of error handling until later.

2.2 and Beyond, cont’d

- Scenario details:
  - *o* Typical scenario shows user selecting some operation.
  - *o* Start with "... the user selects ...".
  - *o* Show resulting screen shot.

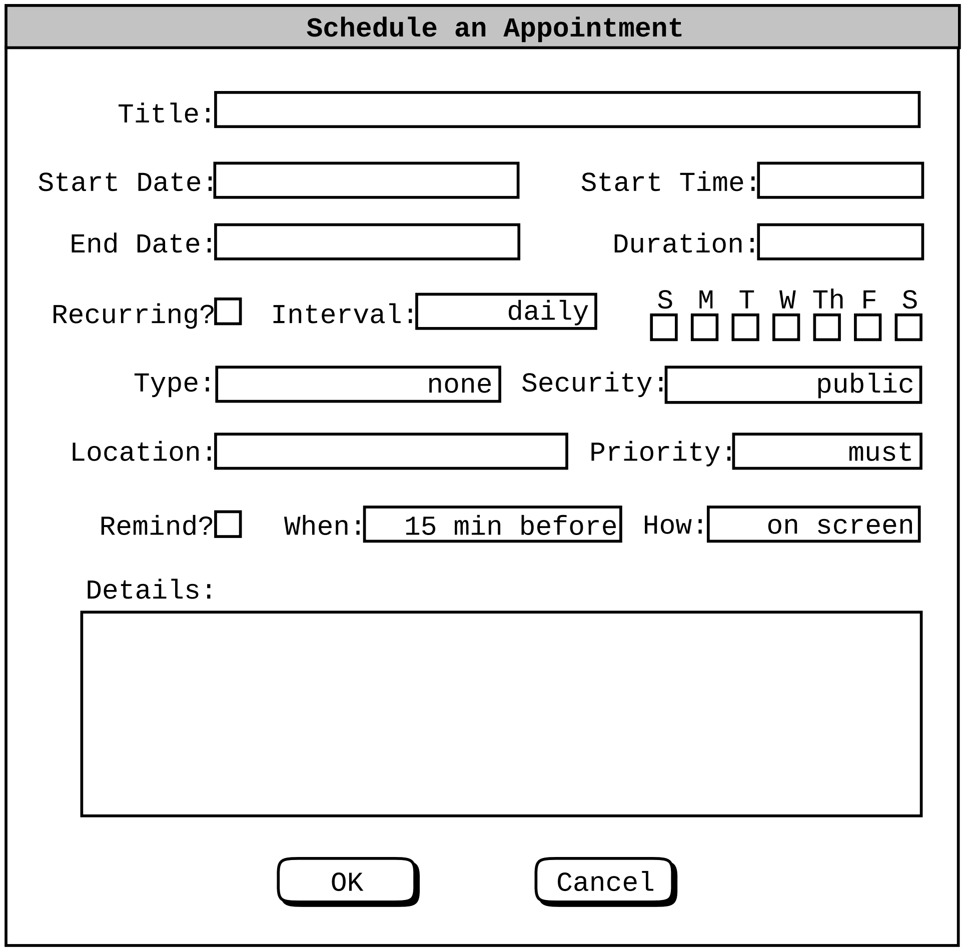
2.2 and Beyond, cont’d

- Explain screen contents in follow-on narrative.
- Continue in this style, showing user action and results, with generous explanatory narrative.

Section 2.2: Scheduling Appointment

- This Calendar example is a typical rough draft.
- Figure 3 shows result of selecting ’`Schedule->Appointment`’.
- Explanatory narrative follows.

… user selects `Appointment`


Schedule an Appointment

Title:

Start Date: Start Time:

End Date: Duration:

Recurring? ☐ Interval: daily S ☐ M ☐ T ☐ W ☐ Th ☐ F ☐ S ☐

Type: none Security: public

Location: Priority: must

Remind? ☐ When: 15 min before How: on screen

Details:

Figure 3: Appointment Scheduling Dialog

Scheduling Appointment, cont'd

Typical explanatory narrative following screen:

The title field is a one-line string that describes the appointment briefly. The date is the date on which the appointment is to occur. ...

Scheduling Appointment, cont’d

- Figures 4-7 show results of additional user actions.
- Explanatory narrative interspersed between each screen shot.

Scheduling Appointment, cont'd

... user selects `Category` ...

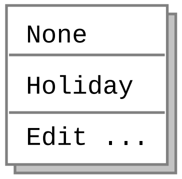


Figure 4: Initial categories menu.

Explanatory narrative ...

Scheduling Appointment, cont'd

... user selects the `'Edit ...'`

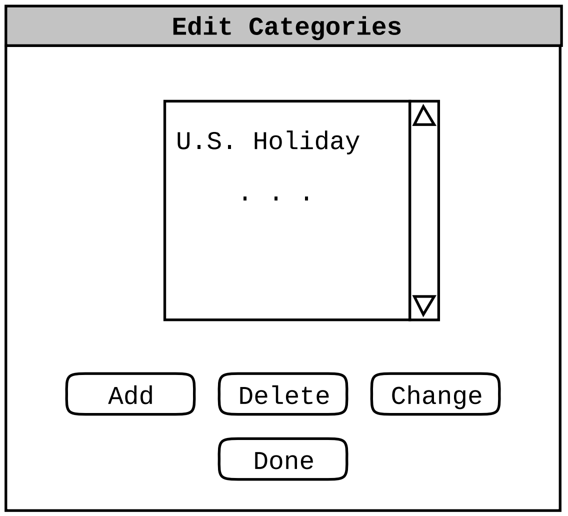


Figure 5: Edit categories dialog.

Explanatory narrative ...

Scheduling Appointment, cont’d

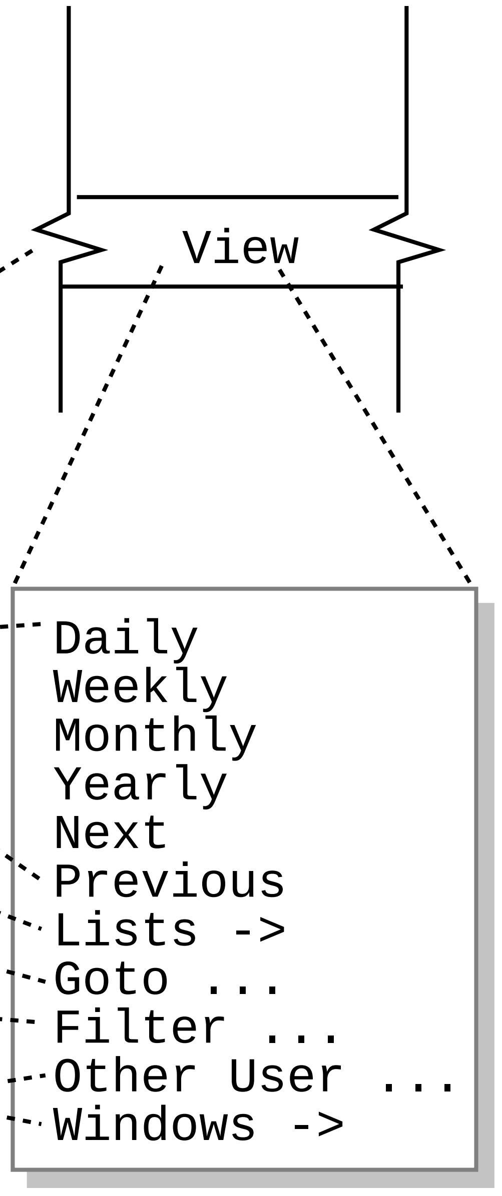
- The *explanatory narrative* ... parts will become more refined.
- Eventually, all commands and data formats are covered at least once.
- We’ll discuss further details of scenario style in upcoming lectures.

Section 2.3. Browsing

- Editorial remark explains that this and remaining sections are skeletons.

- A number of browsing scenarios are planned.

- Scenario order generally follows layout of commands in ’`View`’ menu.



2.3 Browsing an Indvidual Calendar

2.3.1 Viewing by Day, Week, Month, and Year

2.3.2 Viewing Lists of Scheduled Items

2.3.3 Viewing a Specific Calendar Date

2.3.4 Filtered Viewing

2.3.5 Viewing Recently Displayed Windows

2.3.6 Viewing Other Users’ Calendars

2.3.7 Receiving Reminders

Critique of Section 2.3 Rough Draft Organization

- For consistency, use term "Viewing" instead of "Browsing".
- Section 2.3.1 may get too big.
- Flip order of 2.3.5 and 2.3.6 to be consistent with functional hierarchy.
- Minor details at this point, but worth noting.

Section 2.4. More Scheduling

- These scenarios cover remaining commands in '`Schedule`' menu.
- Stylistically, the "simple-to-more-detailed" guide-line is being used here.
  - I.e., start with simple scenario on basic sched-uling (Section 2.2).
  - Cover remaining details subsequently.

Section 2.5. Scheduling Group Meetings

- This scenario covers scheduling from a group leader’s perspective.

- Stylistically, the "user-category" guideline is being used here.

  - I.e., start with scheduling scenario for most common user category (registered user).

  - Present a subsequent advanced scenarios.

Section 2.6. Admin Functions

- Scenarios ’`Admin`’ menu commands.
- Stylistically, things come together naturally here.
  - *o* Follow the functional command hierarchy.
  - *o* Commands for different user category (admin).
  - *o* Somewhat mundane operations towards end.

Sections 2.7 and 2.8. Options, File, Edit

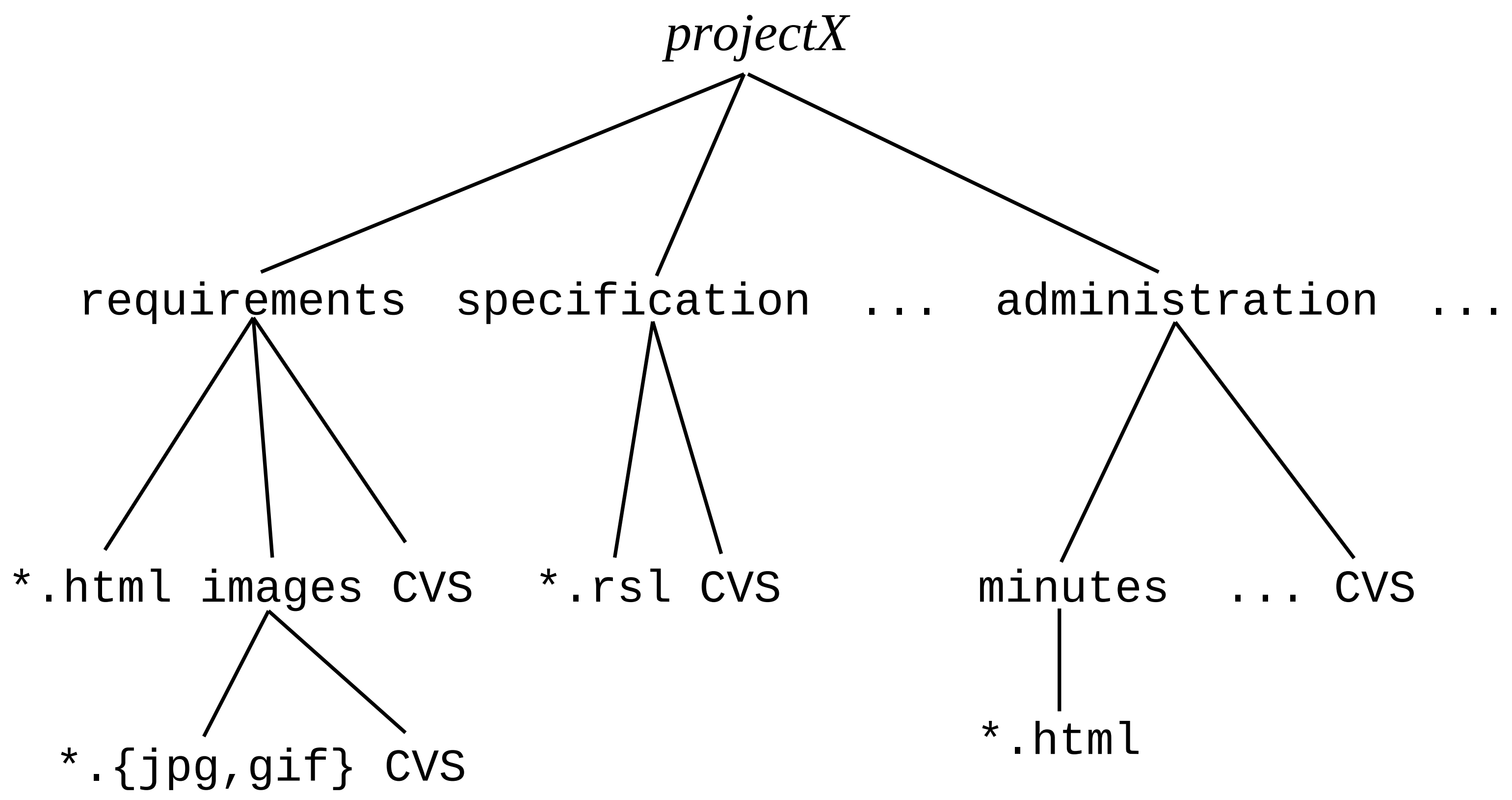
- Again, we’re following the "mundane details towards end" guideline.
- These details are important, but not what the Calendar Tool is mainly about.
- The point is, we try to keep the reader engaged without compromising overall organization.
- Use your own good judgment for your projects.

Where Things Stand with Milestone 2

- A very rough draft.
- Focus on fundamental functionality.
- Error conditions not yet considered.
- Much work yet to do.

SOP Volume 1

Project Directory Structure



Specific Update Procedures

- Each project member (including librarian) has her/his own *work* directory.
- There is a master *projects* directory maintained by the project librarian.
- See Figure 2 in handout.

Update Procedures, cont'd

- Changes originate in individual work directories.
- Team members checkin their work using
  svn add and *svn commit*.
- Team members checkout colleagues' work using
  svn update.
- Librarian releases to project directory using
  svn update.

Update Procedures, cont’d

- Check in happens at least weekly.
- Individuals check in their work.
- Librarian "releases" to public project directory.

File Ownership

- Exactly one member owns each file.
- Only owner checks in.
- Other members check out.
- Ownership recorded in file
  ```
    administration/
         work-breakdown.html
  ```

SVN Basics

- SVN is "Subversion" version control tool.
- It maintains a version *repository* that records the history of a project’s files.
- Members of a project team each maintain an individual *working* directory.

SVN Basics, cont'd

- There are two fundamental operations of any version control system:
  - *o* file *check in*, from a individual working directory to the repository
  - *o* file *check out*, from the repository to a working directory

SVN Basics, cont'd

- In SVN, check in is accomplished using the *svn add* and *svn commit* commands.
- Check out is done most frequently with the *svn update* command.

SVN Basics, cont'd

- Other useful SVN commands include:
  - *o* removing unnecessary files
  - *o* checking file status
  - *o* controlling which files are put in repository
  - *o* comparing past versions

- SVN basics handout covers details.

SVN Basics, cont’d

1. **Initial library setup**

    Done by librarian one time only.

SVN Basics, cont’d

2. Initial project checkout

```
cd
mkdir work
cd work
svn checkout file:///home/librarian/
   your-project/projects/SVN/trunk/your-project
```

Performed one time only.

SVN Basics, cont’d

3. Checkin new work

```
cd ~/work/your-project/...
create some-file
svn add some-file
svn commit -m "log message" some-file
```

Performed the first time you check in a file.

SVN Basics, cont'd

4. Checkin revised work

```
cd ~/work/your-project/...
edit some-file
svn commit -m "log message"  some-file
```

Performed every time you revise a file.

SVN Basics, cont’d

5. Checkout team members’ work

```
cd ~/work/your-project
svn update
```

Performed to get your teammates’ latest work.

SVN Basics, cont’d

6. Release (by librarian) of team work

```
cd ~librarian/projects/work/your-project
svn update
```

Performed by librarian to hand in group’s work.

SVN Basics, cont’d

7. Removing previous checked in files

To remove file named "*X*" from repository:

```
svn remove -f X
svn commit -m "log message"
```

Performed to remove a file from the repository.

SVN Basics, cont'd

8. Viewing status

```
cd ~/work/your-project
svn status -u
```

Produces file list with the following status codes:

SVN Basics, cont’d

Code	Meaning
M	**M**odified file, i.e., you’ve made some changes and need to commit the file.
?	Unknown file, need to add and commit it.
!	UNIX rm’d file wihtout svn remove.

SVN Basics, cont'd

Code	Meaning
A	**A**dded file via '`svn add`', needs to be committed.
R	**R**emoved file via '`svn remove`', needs to be committed.
C	**C**onflict exists (see below for details).

SVN Basics, cont’d

- If ’`*`’ appears, team member has made changes.
- If both ’`M`’ and ’`*`’, conflict exists -- see below.

SVN Basics, cont'd

9. Differencing Modified Files

For any file *X*,

`svn diff` *X*

diffs working and repository copies.

SVN Basics, cont'd

10. Viewing a log report

For any file *X*,

```
svn log X
```

or for an entire directory recursively, just

```
svn log
```

SVN Basics, cont’d

11. Undoing Working Changes

For added or removed file *X*,

```
svn log X
```

undoes add or remove.

Also erases local uncommitted changes.

SVN Basics, cont’d

12. Dealing with a Conflict

For conflicting file X,

```
mv X X.sav
svn update X
```

Then compare X with X.sav to see how to deal with the differences.

SVN Basics, cont’d

13. Telling svn to ignore certain files

In the directory where the files to be ignored reside, add file names into `.svnignore` file. Then

```
svn propset svn:ignore -F .svnignore .
svn commit -m "Ignored files ..."
```

SVN Basics, cont'd

14. Connecting to a SVN server remotely

- Install `svn` and `ssh`, if necessary.
- Run

`svn checkout svn+ssh://`*id*`@vogon/home/`*librarian/*
   your-project`/projects/SVN/trunk/`*your-project*

  - Use command line or GUI client.
  - See Lab Notes 3 for more details.