Momento

Team 3
Christine Ryu
Dahlia Bock
Faustinus Gozali

Overview

- What is Momento?
- What were our goals?
- What did we build?
- System Flow
- Demo

Overview: What is Momento*?

- Mobile Messaging and Evaluation Tool
- Facilitate early stage development and testing for mobile applications
- Developers can simulate and study mobile apps without device specific development
- Has already been installed on a desktop computer

*Information taken from *Momento: Early Stage Prototyping and Evaluation for Mobile Applications*, Scott Carter, Jennifer Mankoff

Overview: What is Momento?

- How does Momento relate to our project?
  - Example: Desktop can send messages to survey users

Overview: What were our goals?

- Create a GUI for a mobile device, i.e. cellphones to support Momento messaging
- More specifically, we want to service all messages sent by Momento automatically

Overview: What did we build?

- The GUI for a cell phone using J2ME

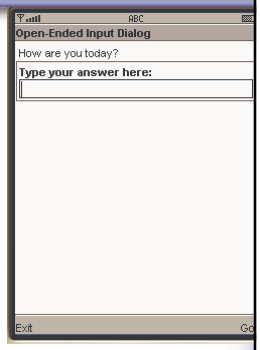
Overview: What did we build?

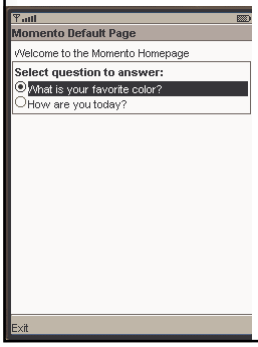
- UI Template for each type of question/SMS
  - Multiple Choice
  - Single Choice
  - Open Ended
  - Numeric
  - Yes/No

Overview: What did we build?

The system is separated into 2 parts

Application is launched automatically by the Push Registry when a message arrives





The user can manually launch the application to view unanswered messages

Overview: System Flow

- Flow for when activated by push registry:
  - 1. Desktop application sends a message to a cellphone□
  - 2. Push Registry activates the application
  - 3. Loads Unanswered Queue from Record Store
  - 4. Message is parsed into a Question object
  - 5. Corresponding UI is displayed
    - user is prompted to answer message
  - 6. If answered, response sent to database through HTTP request. Else, message added to unanswered queue.
  - 7. If new message has arrived, go to step 4
  - 8. When exiting, Unanswered Queue is saved to Record Store

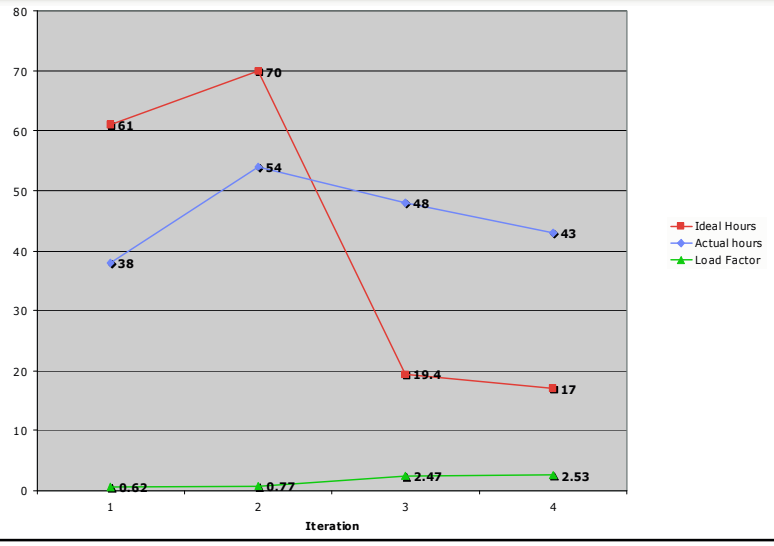
Overview: System Flow

- Flow for when activated by user
  - 1. Load unanswered queue from Record Store
  - 2. Display Welcome/History page
  - 3. Display selected Question
  - 4. If answered, send response to database, remove from unanswered queue, and return to step 2. Else if exited, add to unanswered queue and exit application
  - 5. Before exiting, save unanswered queue to Record Store

Overview

- Demo…
  - Desktop sends message
  - Desktop sends another message
  - User answers 1, and exits another
  - User enters welcome/history page
  - Answer the listed question

Iteration Actual/Ideal Hours and Load Factor



Original Picture of Success

- The application will display correct UI on emulators for each type of questions (open, multiple choices, single choice, yes or no, and numeric). The UI design must meet the criteria as agreed by us and the client.
- The application is able to accept and parse text input, and display corresponding UI to emulators. The application will handle invalid input robustly.
- User's response to the questions posed is sent out and saved to an external MySQL database.
- Successfully transfer UI from emulators onto a particular mobile device (cellphone).
- Notification feature for user who has not responded to question after a certain time period *t* is implemented
- Persistency feature; for example we want to obtain user input every 2 hours over the course of the day.

What requirements changes occurred?

- J2ME Polish
  - Spent a lot of time reading APIs but documentation was sparse and unclear
- Notification
  - Became downgraded to become an extra feature after one group member dropped the class
- Persistency
  - Was an extra feature but client wanted an implementation, so a basic one was provided

What design changes occurred?

- Threading
  - Added a MessageQueuer component to our design to accept messages in the background
  - Initially assumed that only one message will arrive at a time
- Numeric message type
  - Added a parameter to allow different answer sets
- Message format
  - Added *userid* and *questionid*

What did we do well?

- Able to deal with user stories that required more time than estimated
  - E.g. J2ME Polish, message queueing
  - Approached our client to revise user stories

What did we do poorly?

- Overestimation of time and effort
  - Was not familiar with environment
  - Caused our load factor to be skewed
  - Learned better toward the last 2 iterations

Other lessons learned

- XP praise
  - We appreciated the flexibility of user stories and communication between us and our client
- XP criticism
  - Pair programming: really hard to get people together, was not more effective than two people programming alone

Conclusion

- Enjoyed the project and appreciated flexible client communication (Thanks Scott!)
- Was exposed to XP programming practices
- We hope that this project will be useful