System Software - GUI Applications

Videos

https://youtu.be/M8PaixAINwY - Lect-24-2150-pt1.mp4
https://youtu.be/o8c6_0g21cg - Lect-24-2150-pt2.mp4
https://youtu.be/hLVi4QN-ZeM - Lect-24-2150-pt3.mp4
https://youtu.be/JpvLyZ1db94 - Lect-24-2150-pt4.mp4
https://youtu.be/uzW_vfFisKI - Lect-24-2150-pt5.mp4

From Amazon S3 - for download (same as youtube videos)

http://uw-s20-2015.s3.amazonaws.com/Lect-24-2150-pt1.mp4
http://uw-s20-2015.s3.amazonaws.com/Lect-24-2150-pt2.mp4
http://uw-s20-2015.s3.amazonaws.com/Lect-24-2150-pt3.mp4
http://uw-s20-2015.s3.amazonaws.com/Lect-24-2150-pt4.mp4
http://uw-s20-2015.s3.amazonaws.com/Lect-24-2150-pt5.mp4

Outside Reading

https://nora.codes/post/stop-making-students-use-eclipse/

System Software - Batch Processing Code

Most software from the 1960s to the 1990s for business ran on a “bach” processing model. It worked a lot like an assembler - but in this case it worked on financial data.

Banks still use this - the IRS still uses this. If you get a mortgage - then the bank will use software that works this way. Telephone companies and your utility company still work this way.

Often a database persists some of the data - but the basic model is:

1. Read in a record of data
    - Validate the record
    - Do some processing on the record (save in a database, update a database)
    - Produce some output for this record
    - Summarize this data - for final reports
2. Move to next record (back to (1))

System Software - with GUI interfaces (Microsoft Excel, Photoshop etc)

The 1980 to 2000 saw the development of “desk top” applications like Excel and Photoshop. The are what-you-see-is-what-you-get applications with a GUI interface. You click and things change. A large number of compiled languages were built during this time to reflect the difficulty in developing these applications. Often the GUI interface in these is 90% or more of the code in the entire application. Also applications moved from the Mainframe (batch processing) to the Desktop (GUI with a mouse).

Good Side: GUI is really good for drawing and fast changing of data. This is fantastic for individual creativity and one-off operations.

Bad size: There is no way to re-use or build a higher-level abstraction with a GUI. If I am processing an image to add a watermark with Photoshop. Then it takes about 21 clicks to open image, open a layer, drag the “water mark” layer on top of the image, combine the lawyers, save the resulting image. This is all good when you have 1 image to work on. When you have 100,000 images a week this is a huge waste of time. The same task cam be implemented in shell-script (programming language) in 8 lines of code and do all 100,000 in the same time as it takes a human to do 4 images. Also the shell script will get the “water mark” in the same correct location every time, not in a variable location.

Good Side: Very little training is required.

Bad Side: The code is really really hard to test. This means that there are lots of defects left in it.

Good Side: Well designed applications are a joy to use.

Bad Side: Most GUI applications are very poorly designed. Figuring out how users actually work is incredibly hard. Apple develops 9 different to finish interfaces for every change that they make to a user interface then picks the best one. That is very costly and most business apps don’t have that luxury.

Good Side: it is quick to make changes.

Bad Side: The application is “real time”. Anything longer that about 10ms in response to a mouse move or click makes the interface very hard to use. Not everything that we want computers to do is can be done in 10ms. This introduces concurrency into the code - and that is very hard to work with.

```
  Start a loop waiting for an event.
          On Event – Propogate the change to the region of the screen handler for that ev
```

```
Given the context (region) of the vent – then take action to update the
Redraw the screen based on the changed data in the model.
```

The real difficulty in this is that every region of the screen can now do changes to the “model” for every mouse / keyboard or other event in any order. Every chunk of code also has to make the changes in real time. Also users frequency make mistakes and want to un-do the changes that they just made.

System Software - the “finder”

The universal tool that is GUI based is the “finder” on mac and the “explorer” on windows.

What is the “finder”?

Example of using the finder.

The command line alternative to the finder.

More modern GUI applications

The web is the replacement for most desktop applications. This moves the GUI interface to a client and the processing to a server. This has all sorts of advantages - but it also makes the development more distributed. The biggest advantage is that the users will not install any software on a local system to get it to work. The biggest disadvantage is security.

The last lecture on web-lecture is on how they work.

System Software - Databases

The NASA Saturn V project saw the development of a truly new kind of system software. This was the IBM Information Management System. This was the first “database” system ever developed. IBM followed this with a number of years of research and finally developed a “query” language that allows for the concise storage and retrieval of data. This was standardized and is called SQL for Standard Query Language. It has it’s problems but usually works fairly well.
One of the people, Larry Ellison, that worked at IBM on this left and purchased a small software company, renamed it Oracle and started producing a database based on this.
The two most important storage systems for data are Microsoft Excel, and the Oracle Database.

First thing is that databases store the data in a common location - and - the software remotely accesses the data. This means that the data is not local to the user and the database has to support

a client-server architecture. Most of the web's client server system design came from the building of databases. Being client-server this means that the database has to have security that works. Some systems, Oracle, PostgreSQL have succeeded at this. Some like Microsoft SQL-Server(T-SQL) have known problems that have persisted in security for years. Also applications that run on top of these have to be security-aware and prevent things like malicious code injection.

Database have evolved and de-evolved. In the 1990s SQL and relational took over from the Network and Graph databases because it could do everything that they could do - and do it better and faster. Now in the 2020s we see a resurface in Network (Neo4j) and Graph databases (Mogo DB). My testing indicates that everything that you can do in Neo4j and MogoDB you can do in PostgreSQL at about 4 to 10 times faster - and - you can also do rational queries and a bunch of other things. There are some special applications like Time Series databases that work really well. PostgreSQL has a TimeSerise package - that appears to me to have more features and be the best time series database in existence. There are special databases like Google F1 built on top of BigTable. This is a relational database running on a distributed block share. It uses atomic clocks for synchronization of data. It is the worlds first "world scale" database. This has some really fantastic uses like "gmail" and google "docs". There is an open-source version of the same technique called Cockroach DB that is a port of PostgreSQL to use the distributed block share and not rely on atomic clocks. There are special network databases that Facebook uses. This is called Cassandra. At scale it makes sense. Scale being over 5,000,000 computers running it at the same time. Most people don't need 5 million computers. One of the non-relational databases that I use is called Redis - it is an in-memory database that can easily do 250,000 transactions per second on a single box and - it can scale to 3 or 4 million transactions with replication. This is a fantastic way to build data-queues and caching. I also recognized it's limitations. I don't try to store user-accounts and accounting records in an in-memory system.

Copyright