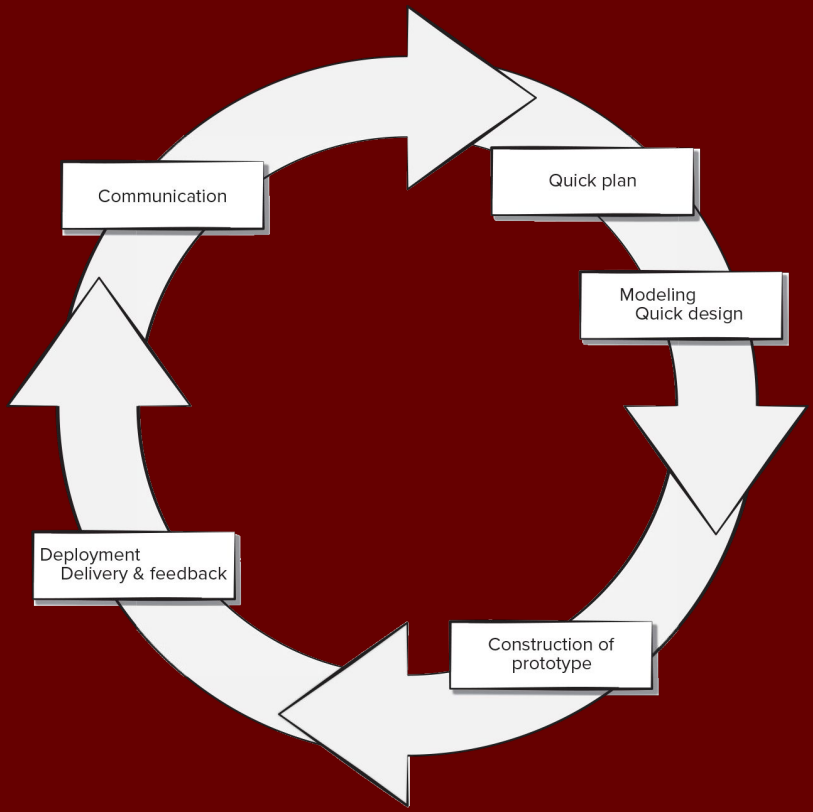
Doctor-R-Us

By Joshua Bryan and Anthony Randall

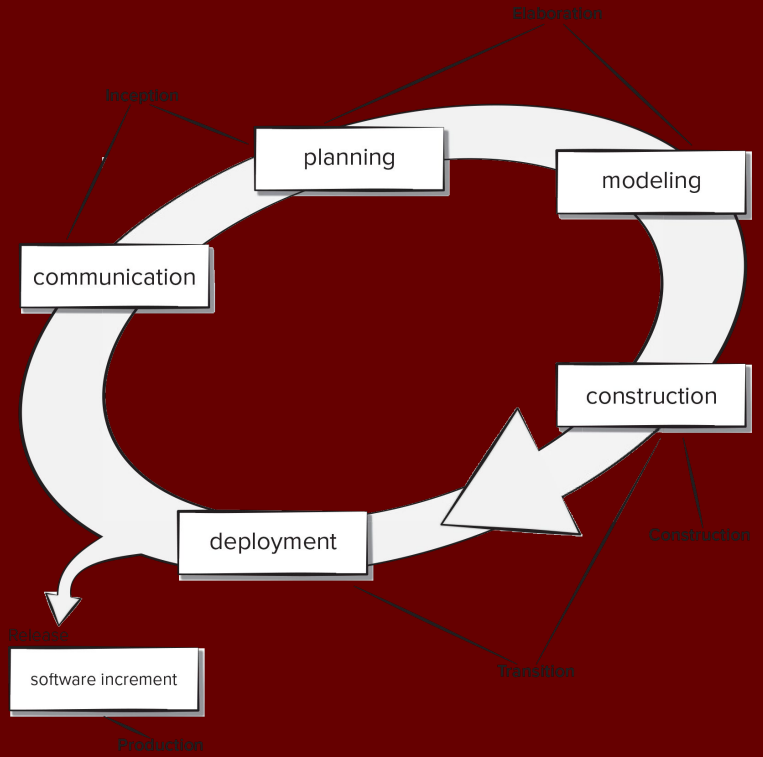
What is the Prototyping Model?

The Prototyping Model is a process model is a software development process where a preliminary version (prototype) of a system is built quickly to visualize and test key features. It allows for quick delivery, adapts well to changing requirements, and encourages high user involvement. Users can provide feedback early, helping shape the system to meet their needs. However, it can lead to scope creep, where constant changes delay progress. Prototypes are often incomplete, leading to misunderstandings about the final product. Additionally, this model is less suitable for large-scale projects due to the challenges of managing multiple iterations.



What is the Unified Process Model?

The **Unified Process Model (UP)** is a common iterative approach to software development. That flexibility is important in today's software world, which is why older models like Waterfall are becoming less utilized. The UP highlights the importance of continuous communication with end-users and allows for the systems architects to focus on key objectives such as system understandability, adaptability to future changes, and the reuse of components.



Characteristics:

		Unified Process Model	Prototyping Model
1	Great for long term development where multiple revisions will be needed over time	✔	
2	Each step is checked over and completed, with efficiency.	✔	
3	Refines the use cases before starting to code the project.	✔	✔
4	Continuously collaborates with stakeholders.	✔	✔
5	Completes larger projects in a efficient manner.	✔	

The Winner?

We ultimately decided that the **Unified Process Model** would be the best fit for our system. This model works well with the way the project had developed. Since the requirements weren’t clear from the start, the cyclical approach allowed us to improve and refine the system’s needs after further discussions with our end user (Professor Engel). This way, we could regularly check in with the end-user, make adjustments, and fine-tune things without having to start over each time, saving us time and effort. Since there were no strict time constraints, we had the flexibility to take our time and ensure we included all necessary features. The project's size also made complexity less of a concern, so managing a few iterations wouldn’t be overwhelming.

Use Case 1: Appointment Scheduling

Who has access? The Receptionist, The Doctor

Description: The receptionist or doctor will schedule an appointment for the patient. The system will then store the patients information including the time of the appointment, the doctor they will be seeing, the reason for their visit, etc.

The receptionist will select the “Schedule Appointment” option in the system.

The system will ask the user to input the patient's name, doctor of choice, date, time, and reason for visit.

The system then verifies if there are any scheduling conflicts, and changes status to “upcoming”..

The system then stores the appointment info and displays a confirmation message.

Use Case 2: View Appointment Information

Who has access? The Doctor, The Receptionist

Description: The doctor can view the scheduled appointment details, this may include the patients details, the reason for the visit, and any past appointments the patient has had, so they can prepare accordingly for the consultation.

The receptionist/ doctor will select the “View Appointments” option in the system.

The system will list the upcoming appointments for that current day

The receptionist/ doctor will select the appointment to view then system will show patients information

The receptionist/ doctor reviews the appointment details for consultation preparation or reference / managing.

Use Case 3: Patient Check-in & Check-out

Who has access? The Receptionist

Description: The receptionist will log when a patient checks in and out. This will help to track and monitor whether an appointment is skipped or cancelled (The user can also update the time attended if forgotten).

The receptionist will select the “Check-in” option for the patient upon arrival.

The system will store the time when user presses “Check-in” as start time. Once patient leaves, receptionist will press “Check-Out” option.

The system then stores the “Check-out” time and changes the status of the patient to “Completed”.

The system will then calculate the duration of the appointment and store it then display a “success” message for the user.

Class Diagram:

Patient

- ❏ patientID: int
- ❏ name: String
- ❏ address: String
- ❏ phoneNumber: int
- ❏ age: int
- ❏ sex: String
- ❏ email: String
- ❏ dob: String
- ❏ insuranceInfo: String

Doctor

- ❏ doctorID: int
- ❏ Name: String
- ❏ Specialty: String
- ❏ emergencyContact: int

- ★ makeAppointment()
- ★ viewSchedule()
- ★ changeAppointment()
- ★ cancelAppointment()

Receptionist

- ❏ receptionistID: int
- ❏ name: String

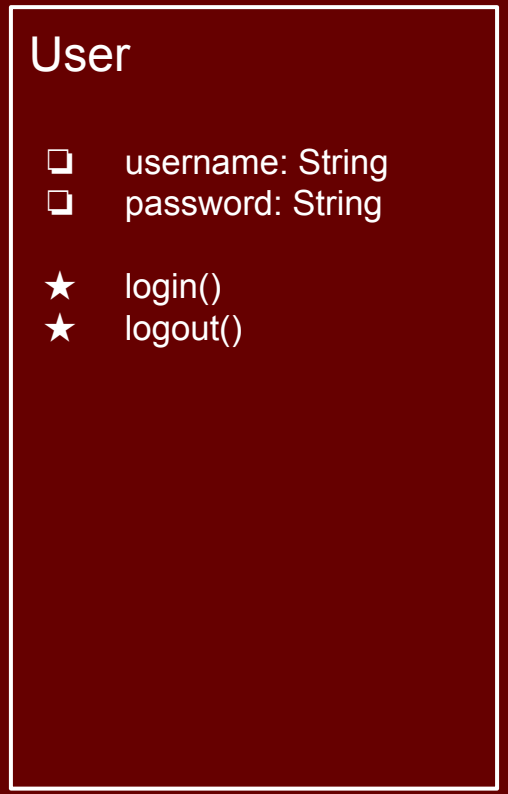
- ★ createAppointment()
- ★ cancelAppointment()
- ★ changeAppointment()
- ★ checkInPatient()
- ★ checkOutPatient()

Appointment

- ❏ appointmentID: int
- ❏ date: String
- ❏ timeIn: String
- ❏ timeOut: String
- ❏ doctor: String
- ❏ duration: double
- ❏ reason: String
- ❏ patient: String
- ❏ status: String

- ★ getDuration()
- ★ markAsMissed()

Class Diagram Continued…



Functional Requirements:

1. System requirements:
   - ➔ The system must keep track of all client's names, addresses (2 address lines), city, state, zip code, phone, fax, cell, and email addresses.
   - ➔ The system must be able to store multiple appointments for each patient. An appointment consists of a date, time, doctor, appointment length, and reason for a visit.
   - ➔ The system must be able to store patient check-in time
   - ➔ The system must be able to track check-out time
   - ➔ The system must track when the patient skips an appointment
   - ➔ The system must track when the patient cancels an appointment
   - ➔ The system must be able to store the doctor's' name, emergency number(s), and specialties
   - ➔ The system must be able to store the Insurance information of each patient

2. User Roles

   - ➔ Admin: Can manage doctors' schedules, view and modify bookings, and manage user access.
   - ➔ Doctor: Can set their availability, view, and update their schedule.
   - ➔ Receptionist: Can assist patients with booking appointments, and access the schedules of doctors.

3. Safety

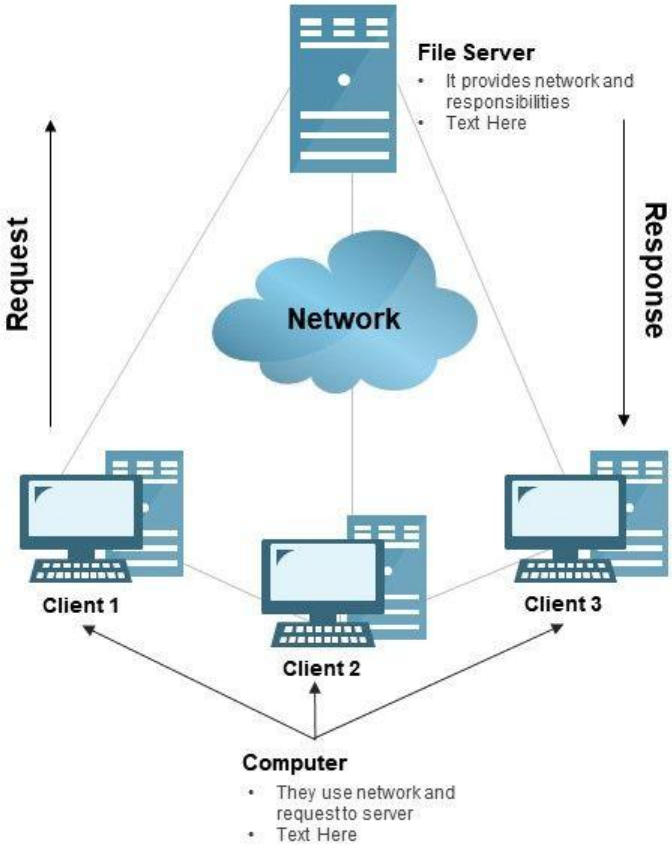
   - ➔ The system should require a two-factor authentication utilizing the user's phone number.

Nonfunctional requirements:

1) Performance
   - ➔ The system should handle concurrent user requests without delays or downtime, particularly during peak hours.
   - ➔ Scalability to support a growing number of users (doctors, patients, and admins).
2) User Interface
   - ➔ Ease of use for both tech-savvy and non-tech-savvy users, by using symbols that are self-explanatory on the website.
   - ➔ Mobile compatibility for users accessing the system via smartphones or tablets.
3) Reliability
   - ➔ The system should be available 24/7 with minimal downtime.
   - ➔ Backup and recovery features to ensure data integrity in case of failures.
4) Compliance
   - ➔ The system should adhere to local medical and data protection regulations (HIPAA).
5) Localization
   - ➔ Support for multiple languages, if the clinic serves patients who speak different languages.

Architectural style

In recent years there has been a huge technological shift towards cloud computing. Utilizing Microsoft Azure as our provider we will be able to delegate the majority of the security protection to the provider which will protect us from being at fault. The benefits to using cloud computing is that there is low latency when it comes to rendering pages and outputs and it can also be opened from anywhere in the world. The biggest advantage is that it is very easy to expand the scheduling shift to other clinics if more are opened up. Connecting to the system will be done through an internet browser and the database will be in the cloud.


File Server
It provides network and responsibilities
Text Here
Request
Response
Network
Client 1
Client 2
Client 3
Computer
They use network and request to server
Text Here

Testing plan:

Following the Unified Process Model, we will conduct a thorough set of tests before the program rollout, including security, performance, usability, and functional accuracy. We'll use white-box to assess the internal logic and structure of the program, while utilizing black-box testing completed by the user to evaluate functionality based on inputs and outputs, while they do not have knowledge of the internal code. When using black-box testing, we will also utilize behavior testing to make sure that inputs are accepted, as well as expected outputs. Additionally, interface testing will ensure proper data input, sequence, and output formatting. This comprehensive approach ensures the software is fully validated for both functionality and performance.

Security plan

Our security plan is going to utilize two factor authentication, microsoft Azure, AES encryption, HTTPS protocols, as well as daily backups to make sure there is no lost information. Two factor authentication will be utilized every time that someone is attempting to access the system, we must make sure that user data is safe at all times. By using Microsoft Azure, AES, and HTTPS protocols we will be able to ensure encryption on all stores passwords, and the transfer of data files to the cloud is secure using internet browsers and Microsoft Azure.

Executive Summary

The Doctor-R-Us clinic upgraded its scheduling system from pen-and-paper to an online, automated solution using the Unified Process Model (UPM). The UPM provided a clear framework, helping us create a thorough testing plan, define functional and non-functional requirements, and develop key use cases. By using a client-server model on Microsoft Azure, we ensured strong security to protect against viruses and malicious threats. Throughout the project, we learned the importance of asking questions and fully understanding the scope, which helped us refine requirements and stay aligned with our goals. This process showed that being thorough and inquisitive early on makes programming easier and results in a more reliable and efficient final product.