```cpp
// Copyright 2017 Patrick Muldoon
#include "intouch.hpp"
#include <string>
#include <iostream>
#include <boost/date_time/posix_time/posix_time.hpp>

Intouch::Intouch(std::string start_line, std::string _filename, unsigned int line_number) {
        filename= _filename;
        success = false;
        start_line_number = line_number;
        end_line_number = 0;
        end_time = "";
        boot_time = 0;


        boost::smatch sm;
        boost::regex_match(start_line, sm, StartRegex);
        start_time = sm[1] + "-" + sm[2] + "-" + sm[3] + " " + sm[4]
                                                 + ":" + sm[5] + ":" + sm[6];
}

void Intouch::BootSuccess(std::string successful_line, unsigned int line_number) {
        success = true;
        end_line_number = line_number;

        boost::smatch sm;
        boost::regex_match(successful_line, sm, SucceededRegex);
        end_time = sm[1] + "-" + sm[2] + "-" + sm[3] + " " + sm[4] + ":" + sm[5]
                                        + ":" + sm[6];

        boot_time = Time_Elapsed();
}

std::ostream& operator<< (std::ostream &out, const Intouch &it) {
        out << "=== Device Boot ===" << std::endl;
        out << it.start_line_number << "(" << it.filename << "): "
        << it.start_time << " Boot Start" << std::endl;

        if(it.success == true) {
                out << it.end_line_number << "(" << it.filename  << "): " << it.end_time
                << " Boot Completed" << std::endl;
                out << "\tBoot Time: " << it.boot_time << "ms" << std::endl;
        }else {
                out << "**** Incomplete Boot ****" << std::endl;
        }

        return out;
}

std::ostream& operator<< (std::ostream &out, const Services &service)
{
        out << "\t" << service.service_name << std::endl;
        out << "\t\tStart: " << service.start_line_number << "(" << service.filename
        << ")" << std::endl;
        if(service.success == true) {
                out << "\t\tCompleted: "  << service.end_line_number << "(" << service.filename
                << ")" << std::endl;
                out << "\t\tElapsed Time: " << service.boot_time << " ms" << std::endl;
        } else {
                out << "\t\tCompleted: Not completed" << "(" << service.filename << ")" << std::endl;
                out << "\t\tElapsed Time: " << std::endl;
        }
        return out;
}

unsigned int Intouch::Time_Elapsed() {
        boost::posix_time::ptime start;
        start = (boost::posix_time::time_from_string(start_time));
        boost::posix_time::ptime end;
        end = (boost::posix_time::time_from_string(end_time));
        boost::posix_time::time_duration total;
        total = end - start;
        return total.total_milliseconds();
}

Services::Services(std::string start_line, std::string _filename, unsigned int line_number)
{
        filename = _filename;
        success = false;
        start_line_number = line_number;
        end_line_number = 0;
        boot_time = "";

        boost::smatch sm;
        boost::regex_match(start_line, sm, ServiceStart);
        service_name = sm[1];


}

void Services::ServiceBoot(std::string successful_line, unsigned int line_number)
{
        success = true;
        end_line_number = line_number;
        boost::smatch sm;
        boost::regex_match(successful_line, sm, ServiceSuccess);
        boot_time = sm[3];
}

Softload::Softload(std::string start_line, std::string _filename, unsigned int line_number)
{
        filename= _filename;
        success = false;
        start_line_number = line_number;
        end_line_number = 0;
        end_time_soft = "";
        boot_time = 0;


        boost::smatch sm;
        boost::regex_match(start_line, sm, SoftLoadBegin);
        start_time_soft = sm[1] + " " + sm[2] + " " + sm[3] + ":" + sm[4] + ":" + sm[5];
        boost::smatch z;
        boost::regex_match(start_line, z, SoftLoadBegin);
        begin = z[3] + ":" + z[4] + ":" + z[5];
}
```

```cpp

void Softload::Originalver(std::string successful_line)
{
        boost::smatch sm;
        boost::regex_match(successful_line, sm, Original);
        oldSoftLoad = sm[6] + "." + sm[7] + "." + sm[8] + "-" + sm[9];
}

void Softload::Newver(std::string successful_line)
{
        boost::smatch sm;
        boost::regex_match(successful_line, sm, New);
        newSoftLoad = sm[6] + "." + sm[7] + "." + sm[8] + "-" + sm[9];
}

void Softload::SoftloadSuccess(std::string successful_line, unsigned int line_number)
{
        success = true;
        end_line_number = line_number;

        boost::smatch sm;
        boost::regex_match(successful_line, sm, SoftLoadEnd);
        end_time_soft = sm[1] + " " + sm[2] + " " + sm[3] + ":" + sm[4] + ":" + sm[5];
        boost::smatch z;
        boost::regex_match(successful_line, z, SoftLoadEnd);
        stop = z[3] + ":" + z[4] + ":" + z[5];
        //boot_time = Time();

}

unsigned int Softload::Time()
{
        std::cout << "breaking here\n";
        std::cout << getStartTime() << " " << getEndTime() << std::endl;
        std::cout << getBegin() << " " << getStop() << std::endl;
        boost::posix_time::ptime x(boost::posix_time::time_from_string(begin));
        std::cout << "broke\n";
        boost::posix_time::ptime z(boost::posix_time::time_from_string(end_time_soft));
        boost::posix_time::time_duration total;
        total = z - x;
        return total.total_milliseconds();
}
```