

Homework 1: HTML and Version Control

For this homework, you create a simple webpage and practice using the basic version control features of Subversion (svn).

You will do this homework as a team; however, each member of your team will be responsible for the completion of a particular task.

Step 1. Checkout a project skeleton

Rather than starting from scratch, you're going to start off with a project skeleton. Each team member must perform this step, so that each has a working copy of the project.

To get a copy of your skeleton, repeat Step 5 (checkout the example web app) from Homework 0 with the following modifications:

- **URL:** https://utopia.cs.memphis.edu/course/comp4081-2013fall/teams/YOUR_TEAM
Where you should replace *YOUR_TEAM* with your teams name (all lowercase; e.g., "falcon"). To test whether you have the correct URL, try entering it into a web browser.
- Fill in the **User** and **Password** fields with login information you emailed me for Homework 0.
- Checkout the **trunk** subfolder of **Homework1**, and name the project **Homework1**.

This time around, you can commit changes you make to the code into the repository. Do so by right-clicking on the project in the **Project Explorer**, and clicking **Team** → **Commit**. A dialog will pop up that shows you what files are being added/removed/updated and that allows you to enter a log message describing what changes you made.

Step 2. Create a web page

Each team member must choose one the web pages in the Tasks section (below) to reverse engineer. All team members must do a different task. If your team has only 5 members, then ignore Task 6.

Each team member must create an HTML5 file in the **src/main/webapp/** directory, and name the file **taskX.html** such that *X* is the number of the task (e.g., **task2.html** for Task 2).

The HTML file you create must look identical to the picture. (Google Chrome is the browser in the screenshots.) Note that your page should use only basic HTML (no CSS, no JavaScript). Be careful not to miss any of formatting details (e.g., italicized text). Each page includes a PNG image, which you can find in the **src/main/webapp/** directory. Do not move or modify these images.

Hint: I expect that your page will include heading, paragraph, bold, image, hyperlink, and list formatting elements.

Hint: Don't forget about the page title that appears in the browser tab.

Step 3. Submit (by tagging) your team's submission

Attention! Before performing this step, you must make sure that all team members have committed their edits to the **trunk** in the repository.

Only one team member (the leader) performs the following.

First, you must fill out the **README.txt** file in your project's **trunk**. The file should list which team member performed each task (one team member per task).

To submit work in this course, you must tag it. Then, I will checkout the revision that you tagged and grade it. By tagging, you tell me that you are done, and this is the version you want me to grade.

The tag you must use for this homework is **hw1** (case sensitive, no spaces).

To tag the current revision of your trunk as **hw1**, do as follows:

1. Go to the **SVN Repository Exploring** perspective in Eclipse.
2. In the **SVN Repositories** view, find the **trunk** folder that you want to tag.
3. Right-click on the **trunk** folder, and click **Show History**. This should open the **History** view with a table listing the past commits to the **trunk**.
4. In the History table, right-click the newest revision (i.e., the one with the greatest revision number), and click **Tag from...** This should open a **Create Tag** dialog.
5. Enter **hw1** into the **Tag** field and optionally enter a log comment, then click **OK**. This should create the tag!

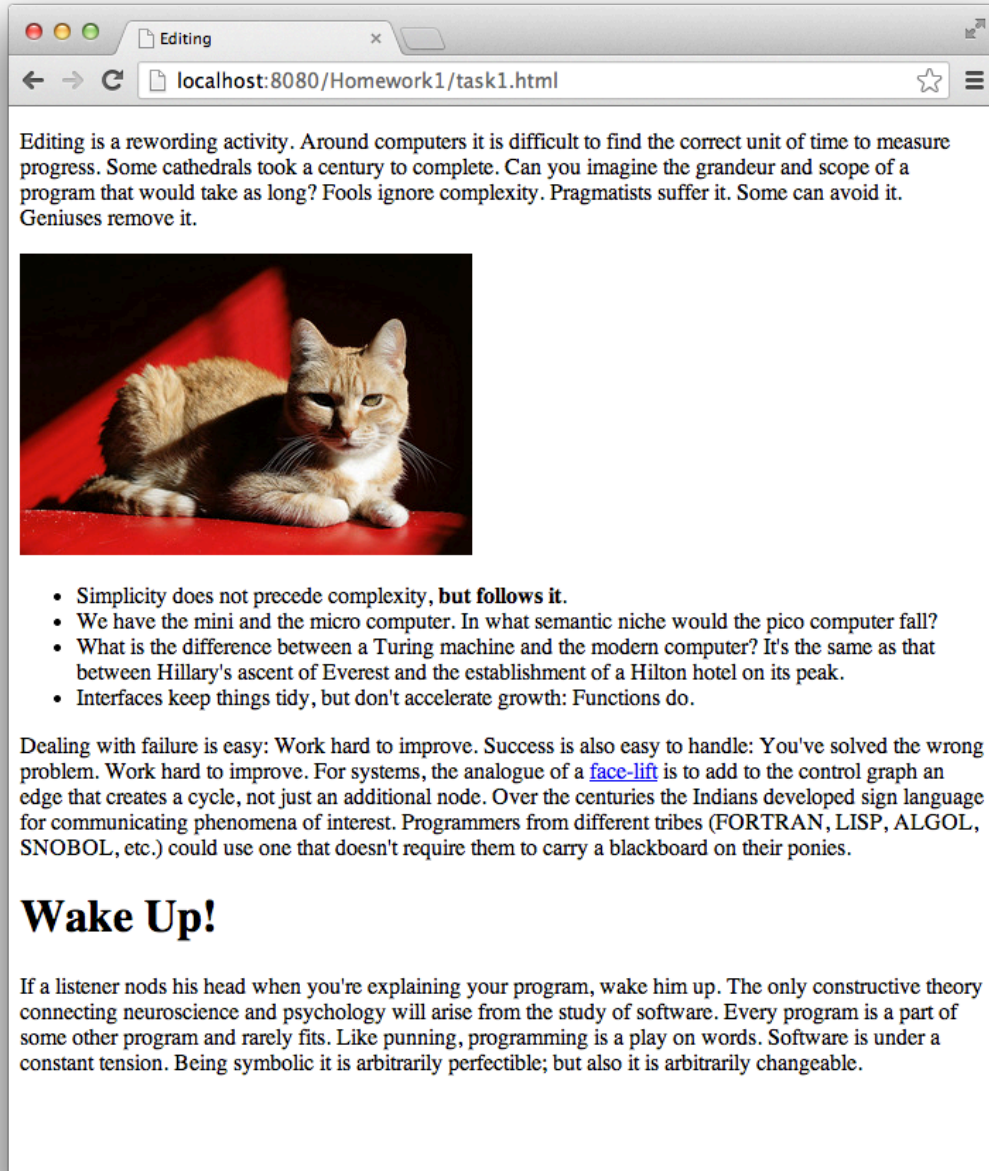
To verify that tagging was successful, open the following URL in a web browser (replacing *YOUR_TEAM* with the appropriate name):

https://utopia.cs.memphis.edu/course/comp4081-2013fall/teams/YOUR_TEAM/Homework1/tags/


You should see an **hw1** folder, and within that folder should be the **src** folder along with the **.project**, **README.txt**, and **pom.xml** files. Everyone's HTML files should be in the **src/main/webapp/** folder.

The Tasks

Task 1



Editing is a rewording activity. Around computers it is difficult to find the correct unit of time to measure progress. Some cathedrals took a century to complete. Can you imagine the grandeur and scope of a program that would take as long? Fools ignore complexity. Pragmatists suffer it. Some can avoid it. Geniuses remove it.



- Simplicity does not precede complexity, **but follows it**.
- We have the mini and the micro computer. In what semantic niche would the pico computer fall?
- What is the difference between a Turing machine and the modern computer? It's the same as that between Hillary's ascent of Everest and the establishment of a Hilton hotel on its peak.
- Interfaces keep things tidy, but don't accelerate growth: Functions do.

Dealing with failure is easy: Work hard to improve. Success is also easy to handle: You've solved the wrong problem. Work hard to improve. For systems, the analogue of a [face-lift](#) is to add to the control graph an edge that creates a cycle, not just an additional node. Over the centuries the Indians developed sign language for communicating phenomena of interest. Programmers from different tribes (FORTRAN, LISP, ALGOL, SNOBOL, etc.) could use one that doesn't require them to carry a blackboard on their ponies.

Wake Up!

If a listener nods his head when you're explaining your program, wake him up. The only constructive theory connecting neuroscience and psychology will arise from the study of software. Every program is a part of some other program and rarely fits. Like punning, programming is a play on words. Software is under a constant tension. Being symbolic it is arbitrarily perfectible; but also it is arbitrarily changeable.

Make the link to: <http://en.wikipedia.org/wiki/Rhytidectomy>


Task 2



The cybernetic exchange between man, computer and algorithm is like a game of musical chairs: The frantic search for balance always leaves one of the three standing ill at ease. When we understand knowledge-based systems, it will be as before -- **except our fingertips will have been singed.**

Build It Top-Down!

Everything should be built top-down, except the first time. We have the mini and the micro computer. In what semantic niche would the pico computer fall? In the long run every program becomes [rococo](#) - then rubble. If a listener nods his head when you're explaining your program, wake him up. When we write programs that "learn", it turns out that we do and they don't.

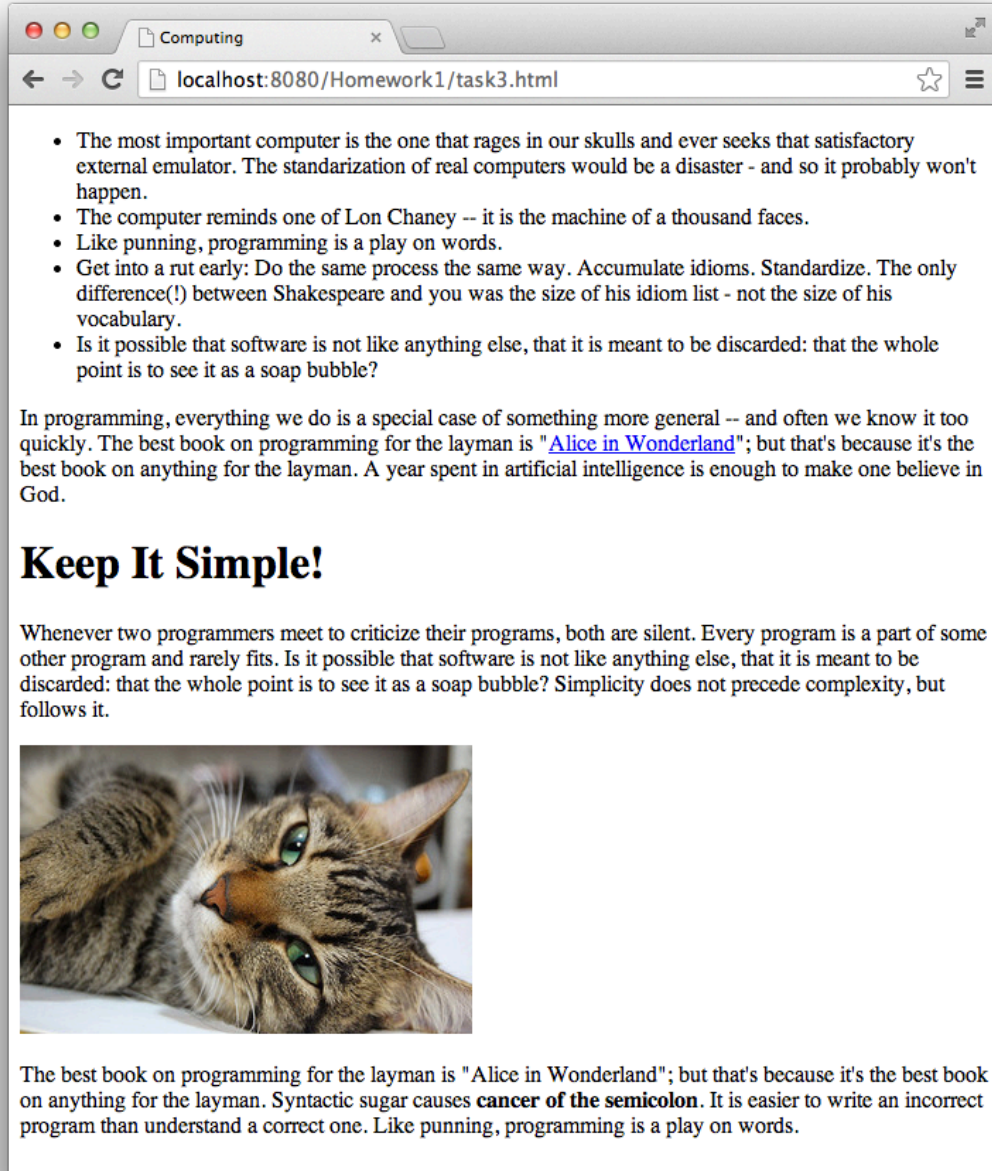


Programming is an unnatural act. In English every word can be verbed. Would that it were so in our programming languages. We kid ourselves if we think that the ratio of procedure to data in an active database system can be made arbitrarily small or even kept small.

1. Documentation is like term insurance: It satisfies because almost no one who subscribes to it depends on its benefits.
2. If your computer speaks English, it was probably made in Japan.
3. Functions delay binding; data structures induce binding. Moral: Structure data late in the programming process.
4. Every program is a part of some other program and rarely fits.
5. The best book on programming for the layman is "Alice in Wonderland"; but that's because it's the best book on anything for the layman.

Make the link to: <http://en.wikipedia.org/wiki/Rococo>

Task 3




The screenshot shows a web browser window with the title "Computing" and the address bar displaying "localhost:8080/Homework1/task3.html". The page content includes a list of five bullet points, a paragraph of text, a bolded section header "Keep It Simple!", another paragraph of text, a photograph of a tabby cat, and a final paragraph of text.

- The most important computer is the one that rages in our skulls and ever seeks that satisfactory external emulator. The standardization of real computers would be a disaster - and so it probably won't happen.
- The computer reminds one of Lon Chaney -- it is the machine of a thousand faces.
- Like punning, programming is a play on words.
- Get into a rut early: Do the same process the same way. Accumulate idioms. Standardize. The only difference(!) between Shakespeare and you was the size of his idiom list - not the size of his vocabulary.
- Is it possible that software is not like anything else, that it is meant to be discarded: that the whole point is to see it as a soap bubble?

In programming, everything we do is a special case of something more general -- and often we know it too quickly. The best book on programming for the layman is "[Alice in Wonderland](#)"; but that's because it's the best book on anything for the layman. A year spent in artificial intelligence is enough to make one believe in God.

Keep It Simple!

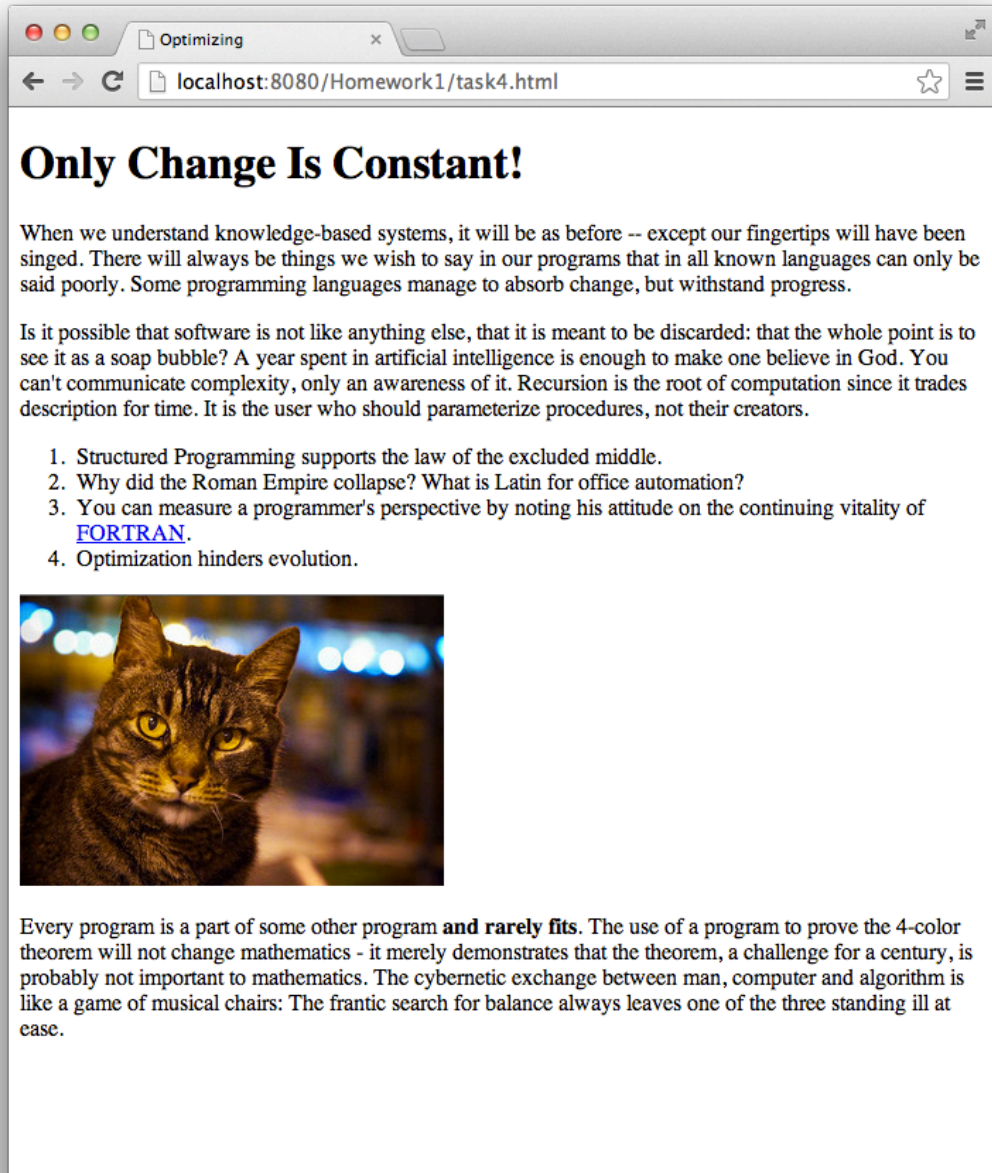
Whenever two programmers meet to criticize their programs, both are silent. Every program is a part of some other program and rarely fits. Is it possible that software is not like anything else, that it is meant to be discarded: that the whole point is to see it as a soap bubble? Simplicity does not precede complexity, but follows it.



The best book on programming for the layman is "Alice in Wonderland"; but that's because it's the best book on anything for the layman. Syntactic sugar causes **cancer of the semicolon**. It is easier to write an incorrect program than understand a correct one. Like punning, programming is a play on words.

Make the link to: http://en.wikipedia.org/wiki/Alice_in_wonderland

Task 4



Optimizing x


localhost:8080/Homework1/task4.html

Only Change Is Constant!

When we understand knowledge-based systems, it will be as before -- except our fingertips will have been singed. There will always be things we wish to say in our programs that in all known languages can only be said poorly. Some programming languages manage to absorb change, but withstand progress.

Is it possible that software is not like anything else, that it is meant to be discarded: that the whole point is to see it as a soap bubble? A year spent in artificial intelligence is enough to make one believe in God. You can't communicate complexity, only an awareness of it. Recursion is the root of computation since it trades description for time. It is the user who should parameterize procedures, not their creators.

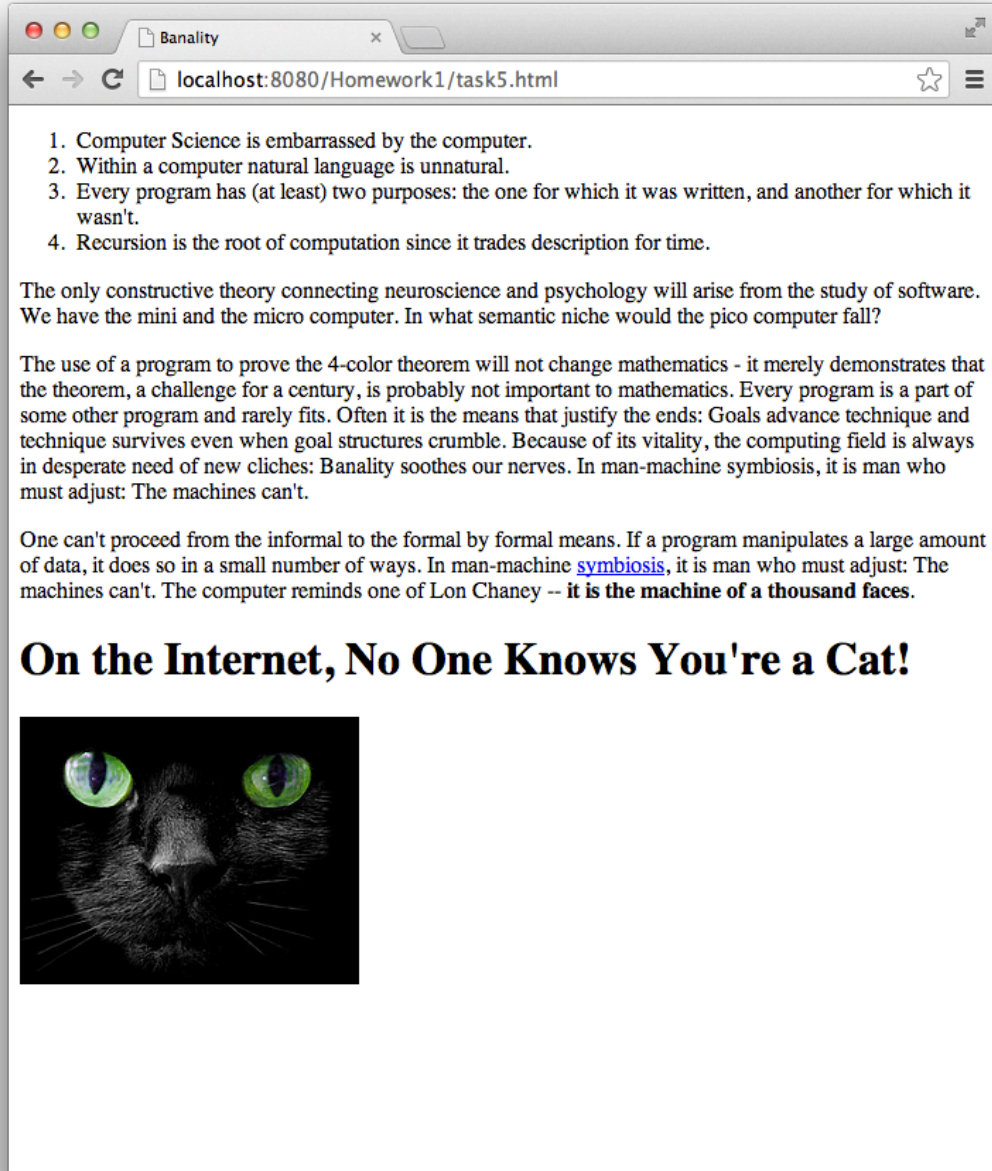
1. Structured Programming supports the law of the excluded middle.
2. Why did the Roman Empire collapse? What is Latin for office automation?
3. You can measure a programmer's perspective by noting his attitude on the continuing vitality of [FORTRAN](#).
4. Optimization hinders evolution.



Every program is a part of some other program **and rarely fits**. The use of a program to prove the 4-color theorem will not change mathematics - it merely demonstrates that the theorem, a challenge for a century, is probably not important to mathematics. The cybernetic exchange between man, computer and algorithm is like a game of musical chairs: The frantic search for balance always leaves one of the three standing ill at ease.

Make the link to: <http://en.wikipedia.org/wiki/FORTRAN>

Task 5



localhost:8080/Homework1/task5.html


1. Computer Science is embarrassed by the computer.
2. Within a computer natural language is unnatural.
3. Every program has (at least) two purposes: the one for which it was written, and another for which it wasn't.
4. Recursion is the root of computation since it trades description for time.

The only constructive theory connecting neuroscience and psychology will arise from the study of software. We have the mini and the micro computer. In what semantic niche would the pico computer fall?

The use of a program to prove the 4-color theorem will not change mathematics - it merely demonstrates that the theorem, a challenge for a century, is probably not important to mathematics. Every program is a part of some other program and rarely fits. Often it is the means that justify the ends: Goals advance technique and technique survives even when goal structures crumble. Because of its vitality, the computing field is always in desperate need of new cliches: Banality soothes our nerves. In man-machine symbiosis, it is man who must adjust: The machines can't.

One can't proceed from the informal to the formal by formal means. If a program manipulates a large amount of data, it does so in a small number of ways. In man-machine [symbiosis](#), it is man who must adjust: The machines can't. The computer reminds one of Lon Chaney -- **it is the machine of a thousand faces.**

On the Internet, No One Knows You're a Cat!



Make the link to: <http://en.wikipedia.org/wiki/Symbiosis>

Task 6



The screenshot shows a web browser window with the title "Failure" and the URL "localhost:8080/Homework1/task6.html". The page content includes a close-up photograph of a cat's face with green eyes. Below the image is the main heading "Failing to Plan Is Planning to Fail!". The text discusses the difficulty of measuring progress in computer programming, using the example of cathedrals. It mentions "Maxwell's equations" and discusses the challenges of planning and dealing with failure. A list of three points is provided at the end of the text.

Failing to Plan Is Planning to Fail!

Around computers it is difficult to find the correct unit of time to measure progress. Some cathedrals took a century to complete. Can you imagine the grandeur and scope of a program that would take as long? In seeking the unattainable, simplicity only gets in the way. If we believe in data structures, we must believe in independent (hence simultaneous) processing. For why else would we collect items within a structure? Why do we tolerate languages that give us the one without the other?

It is not the computer's fault that [Maxwell's equations](#) are not adequate to design the electric motor. It goes against the grain of modern education to teach children to program. What fun is there in making plans, acquiring discipline in organizing thoughts, devoting attention to detail and learning to be self-critical? Dealing with failure is easy: Work hard to improve. Success is also easy to handle: **You've solved the wrong problem.** Work hard to improve.

You can measure a programmer's perspective by noting his attitude on the continuing vitality of FORTRAN. Why did the Roman Empire collapse? What is Latin for office automation? It is not the computer's fault that Maxwell's equations are not adequate to design the electric motor. It is easier to change the specification to fit the program than vice versa. Around computers it is difficult to find the correct unit of time to measure progress. Some cathedrals took a century to complete. Can you imagine the grandeur and scope of a program that would take as long?

- Re graphics: A picture is worth 10K words - but only those to describe the picture. Hardly any sets of 10K words can be adequately described with pictures.
- A language that doesn't affect the way you think about programming, is not worth knowing.
- Every program is a part of some other program and rarely fits.

Make the link to: http://en.wikipedia.org/wiki/Maxwell%27s_equations