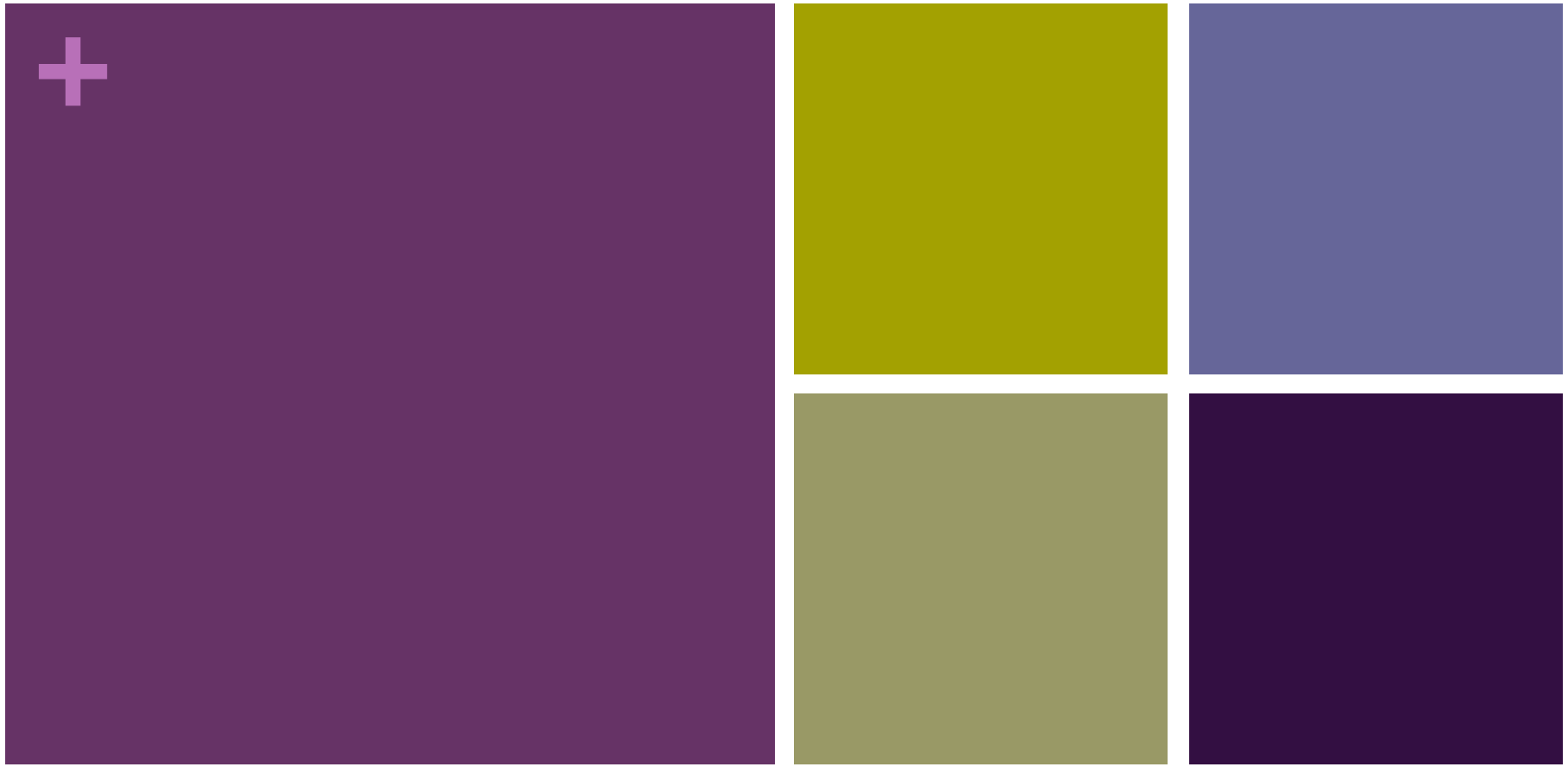

Myths of Correctness

Phathom Donald

Summary

Computer programs were very complex entities that will never be error-free. This is not to imply that the engineers or designers who built the programs are incompetent. Computer programs, or software, because of its flexibility, complexity, potential, and many other factors, will always contain "bugs." They are not subject to the natural constraints that physical objects are faced with. Only maintenance and adequate testing can be applied to these software not to avoid issues, but to manage them.

Software Crisis

- The increasingly apparent difficulty of writing large reliable computer programs led not only to the term "software crisis," but also to a whole field of study called **software engineering**.
- Of all the problems inherent in the software crisis, the most significant is the inability to writer error-free software.

False Election

- During the 1980 election, Jimmy Carter conceded because his pollster warned him that he was going to lose. This warning was prompted from the computer predictions used by TV networks that declared Reagan the winner based on early returns.
- This event exemplifies people's tendency to trust technology and how they rarely question its validity and instead question its effects.
- A similar software crisis occurred in Quebec when an unpopular political party, the Union Nationale, was seemingly leading in an election. However, software bugs were attributing votes to the wrong candidates, and the TV stations did not question it.

Space Shuttle Computer Problems

- On board the U.S. Space shuttle, redundancy is achieved by five identical computers: four of them run exactly the same problem and compare their results during critical flight errors, while the fifth computer operates a program that provides a backup flight-control system.
- A bug occurred when the fifth computer attempted to "tune in" with the other four.
- It arose from combining the primary software with the backup software that exists only in case the primary software doesn't work.

Maintenance

- Running the same program thousands of times may cause the computer hardware components to wear and require maintenance, but the software will cannot wear out.
- A program does not fail because it has worn out, but because it didn't work properly to begin with and that is finally being recognized.
- If software products were never accepted until they were error-free, few companies could ever finish their development contracts.
- Most software products are comparable to lemons found at car dealerships.

Inadequate Testing

- The inadequacy software testing is not the result of incompetence because it is impossible to expose all of the bugs in a program by means of exhaustive testing.
- **Black Box Testing**: to try every operation and see if the program gives the correct results. It is hopelessly inadequate as the basis for any thorough evaluation of computer program correctness.
- No matter how diligently a computer program is tested, it cannot be tested completely.
- However, not to test a program before depending on it would not be recommended.

Flexibility: Good or Bad?

- A computer's behavior can be changed radically by changes to its software. Major changes can be accomplished quickly and at low cost. This makes flexibility seem like a blessing.
- However, software complexity can grow quickly, leading to software that's hard to read, hard to understand, and likely to contain errors. This makes flexibility seem like a curse.
- Computer software lack natural constraints. Each new feature may interfere with several old features and each attempt to fix a bug may create several more.

Invisible Interfaces

- A common approach to writing large programs is to divide the problem into parts and write a separate program for each part.
- Much of the difficulty arises because the separate programs must interact to solve the overall problem.
- The more complicated an interface, the more likely it is that something will fall through a crack. Software interfaces are so error-prone because it is so easy to build complicated interfaces.
- Interface related problems are common in any computer program, but their prevalence grows rapidly with the size of the program.

The Art of Scaling up

- Programmers constantly act as if the skill and effort required to build small computer programs can be scaled up easily to build large programs.
- Problems in some projects arose in large part because the people involved underestimated the difficulty of scaling up their previous efforts.
- Scaled-up software is not only harder to produce, it's harder to maintain.
- When maintaining a program that someone else wrote, your understanding must rely solely on the program's text.
- To manage complexity, appropriate tools are needed such as **support software**.
- To avoid excessive complexity, something analogous to the discipline imposed by the natural constraints on building physical objects in a necessity.
- Complexity cannot be eliminated, but it can be managed.

Lessons to Be Learned

- Software problems are the unavoidable results of the programming languages and methods that we use.