

1. Designing and building reliable software

ptg-global.com

1.1. Summary

If you read most of our papers, you'll see a common thread – the user interface is a fundamental consideration in developing successful software. That is, software that works for people.

In June 2006's edition of Scientific American, there is a fascinating article on how to design dependable software. The fundamental premise of the article is summarised in the callout 'Almost all grave software problems can be traced to conceptual mistakes made before programming started'.

It's certainly true that the article is more about the algorithmic aspects of software, rather than the UI, but the central argument is the same – if the design is flawed, then the software is flawed. The author, Daniel Jackson, argues that

'...more often, the overall design is vague and poorly thought out. As the code grows with the addition of piecemeal fixes, a detailed design structure indeed emerges – but it is a design full of special cases and loopholes, without coherent principles.'

He continues

'...Developers rarely articulate their designs precisely and analyse them to check they embody the desired principles'. But with computers now flying airplanes, driving trains and cars, and running most of the financial, communications, trading and production machinery of the world, society has an urgent need to improve software dependability.'

In the same vein, poor software can't be compensated by a good UI. A good UI only delivers a good application if the underlying software actually delivers on the promise the UI creates.

This paper is a direct summary and paraphrase of the Sciam article and introduces the software design tool, Alloy, developed by researchers at MIT. Alloy checks complex software for conceptual and structural design flaws. This is particularly important for software with numerous objects with numerous states where a particular combination of states may cause the software's downfall. Alloy simulates every state that the software can take to determine that none leads to failure.

Alloy is a free application available at <http://alloy.mit.edu>.

The original Sciam article can be found here

<http://www.sciam.com/article.cfm?articleID=00020D04-CFD8-146C-8D8D83414B7F0000&sc=I100322>

1.2. Software design, reliability and Alloy

Software design has two components to it that are fundamental to its success. The first is the user interface, the second is the architecture. Interestingly, the success of our XPDesign user interface design methodology owes its rigour to software engineering practices.

In the case of software architecture and design, the developer must model the design and then test various permutations to identify any unusual conditions that may cause it to behave in unexpected ways. Tools like Alloy work are predicated on considerable research into methods of mathematically determining whether programs are correct. These methods are exhaustive, in that they test all possible permutations. It is not correct to call them 'smart' in that they can predict the likely areas problem areas and just test those. In other words, they are brute-force methods of application stress testing.

However, what's interesting is that in order to use tools like Alloy to test your application before you code it, you must represent its behaviour at an appropriate level of abstraction. And this is exactly the same process that XPDesign uses when architecting the user interface, before getting to the business of actually designing the layout of a screen.

Let's take a look at how abstraction helps application development and how Alloy helps during the process. The key message here is that you can't just start coding and see how it works as you go. Jackson using the file management as an example and we'll paraphrase and summarise that example here.

Step 1. Define the objects

Here, the application's core objects are defined. In the case of file management, these are 'files' and 'folders', joined by a relationship called 'contains'. That is, a given folder contains a set of given files.

Step 2. Model the operations

Next, the designer models the operations permitted on files and folders.

Files can be moved to any other folder. Folders can be moved to any other folder (i.e. giving the folder a new parent). When a folder is moved, all of its contained objects (other files and folder contained within the folder) move, too.

Step 3. Specify requirements

Whenever a move is executed, it is critical that the new location of the item is in a 'reachable' path from some root directory. That is, if the old location is:

```
C:\program files\some program\preferences
```

And we want to over the preferences item to another location, then that new location is also reachable by starting from the C:\ root. (You may have to think back to the dark DOS days where we used to access files using the command prompt!!).

Step 4. test permutations, then find and fix the flaws

When modelled in Alloy, you can then test permutations to find operations (i.e. 'moves') that cause a conflict.

For example, if, say, your directory structured looked like this:

```
C:\program files\some program\preferences\user 1
                                     \user 2
                                     \user 3
```

Alloy would attempt to move the object 'preferences' (a folder) to the location 'user 2' (a folder contained within the 'preferences' folder). This would be effectively moving the object 'user 2' inside itself. This would (in simple terms) cause a circular conflict that the file management application may attempt to perform an infinite number of times. This would probably lock your system and require a reboot to stop the loop. You would probably lose your folder and files, too.

Therefore, what Alloy is doing is identifying the conditions that would cause problems. You can then write code handle these conditions. For example, the program would then evaluate the location (the target folder) you want to move the folder to. If it is contained within the object you are moving, then it would reject the move and bring up a warning error, preventing the action from taking place.

However, Jackson warns that testing all permutations will not guarantee the application will never crash. Instead, it is a tool to reduce the risk. We'll cover this, next.

1.3. Alloy vs traditional bug testing

Alloy is not analogous to bug testing where a set of input conditions is tested against expected output conditions. In this circumstance, the code is already written and you are determining if it physically works.

Instead, Alloy is about modelling and testing the fundamental design before its coded. The objects and operations are modelled within the Alloy environment before they are coded. The results of testing with Alloy are used to refine the design model. It is about getting the fundamentals right so the application has a robust 'backbone'.

Jackson argues that the 'fixes' applied to the application following testing often exacerbate the problem by causing 'barnacles of complexity'. He uses the analogy of Ptolemaic models of planetary motion. In his model, Ptolemy used epicycles to model the retrograde motion of the planets. As measurement techniques increased, later astronomers realised that his model was inaccurate. To fix it, they first made minor adjustments and then had to add epicycles on epicycles when that also proved inaccurate. Further enhancements also did not solve the problem of accurately modelling planetary motion because the fundamental model was wrong. Using a more modern phrase 'you can't put lipstick on a pig'.

1.4. Improving software design

Alloy does more than provide a testing tool during modelling and design. It helps us design the application in a more systematic and abstract way, rather than the typical urge of most IT managers and developers to get started on the coding as soon as possible.

It mandates a more disciplined approach to modelling the problem to be solved. Essentially, Jackson is arguing for more than the traditional methods such as flowcharting. Alloy is dependent on using an object oriented approach to modelling where you identify the moving parts (objects) and describe their behaviours (methods / operations). Relationships and constraints exist between the objects that ultimately reflect the business rules and processes.

None of this sounds particularly unique, as these techniques are generally well known. However, what we've found is that this disciplined approach doesn't often take place, or if it does, it is at the wrong layer of abstraction. Typically, we see it take place at the data level, not the object level, meaning software ends up being about data, not about work. The lack of modelling that takes place may be because it has not generally seen a direct relationship to software reliability. Now, with an approach like Alloy, the models do have a direct use in testing software behaviour earlier in the development lifecycle.

When testing your application design with Alloy, it would attempt to find scenarios (combinations of states, methods, etc.) that violate the constraints and rules. These can then be investigated by examining the causes. That is, where in your definition of relationships and constraints were you imprecise?

1.5. Alloy, software design and user interface design

Jackson touches on the impact of Alloy's methods on the user interface and usability, suggesting that it would lead to improvements. He concludes the article by saying 'one day, perhaps, software systems will be truly robust, predictable and easy to use – by design'.

We see an incredible parallel between the approach Alloy is taking and how PTG Global engages in the user interface architecture and design approach.

In essence, our XPDesign methodology uses strong psychological techniques to model the domain an application will be operating in from a business and user's perspective. This requires that we use a suitably high level of abstraction to model the core objects the user will work with in the interface and the activities they will perform on them to achieve their job / work goals.

With these core objects and activities, we are modelling the backbone of the application, where all other features and functions 'hang off' this back bone. It provides a robust structure to the user interface's architecture and allows scalability and flexibility.

What does this parallel approach mean? It means that at the same time we model the user interface design, developers can reuse much of the model for the application's design. Both aspects of the application (the UI and the operations) can be tested and verified to work before a single line of code is written.

This is when we start designing for reliability **and** ease of use.

1.6. References

<http://alloy.mit.edu>

<http://www.sciam.com/article.cfm?articleID=00020D04-CFD8-146C-8D8D83414B7F0000&sc=I100322>

2. About the Author

Craig is the founder and Managing Director of The Performance Technologies Group (PTG Global), with over 15 years in user experience, user interface design and change management.

Craig runs the R&D function at PTG, having produced a number of world firsts including XPDesign – the first systematic methodology for user interface design and Certified Usable – the first guarantee for usability and user experience.

Craig has been the primary architect behind many of Australia's most popular websites including CBA, Virgin Blue and ASIC and works on cutting edge technologies such as touch, medical and special-purpose applications.

Craig holds a Masters qualification in organisational psychology, is a member of the APS and the APS College of Organisational Psychologists and is a Registered Psychologist in NSW. He is also an Associate of the University of NSW and Macquarie University.



Contact Craig on:

Email: craige@ptg-global.com

Phone: +61 (0)2 9251 4200

Mobile: +61 (0) 416 266 216

Address: Level 16, 207 Kent St, Sydney, NSW, 2000, Australia