AI Edit Your Code? What the `diff`?

Technical Journal Entry Begins

AI helped me code and I need to see the ***difference*** between the before and after. Thankfully, there's been a program called **diff** around in Unix (and thus Linux too) forever. However, it only lets you see the actual difference between the two files, with this strange plus and minus inserted character syntax. It can be viewed color-coded, which is a great help. The publishing platform I'm using actually supports it with the triple backtick convention that's so popular in markdown. For example:

```diff
--- file1.txt
+++ file2.txt
@@ -1,3 +1,4 @@
 This is line 1
-This line will be removed
+This line was added
 This is line 3
+Another new line
```

...becomes:

```
--- file1.txt
+++ file2.txt
@@ -1,3 +1,4 @@
 This is line 1
-This line will be removed
+This line was added
 This is line 3
+Another new line
```

And that's nice and everything, but it won't show me the whole before and after file for context. And there are plenty of side-by-side file compare tools, but that's not quite what I want either. I just want to see a single color-coded file with the stuff that's been deleted in red and the stuff that's been added in green. Thankfully, git itself comes to the rescue! It can do exactly that, but the output is intended for a color terminal program. So one more step needs to be done to **"pipe"** it into another program, ansi2html using this command:

```
git diff --no-index --color-words -U9999 file1.txt file2.txt | ansi2html > temp.html
```

The output of this is a fully structured HTML file, which takes a little bit of picking apart and deleting chunks in order to embed it here, but it does work:

This is line 1
This line will be removed
This line was added
This is line 3

Another new line

Okay, so that's it. I have my solution. If I get AI coding assistance and I really need to see the before & after to study it, and especially if I need to publish it here, I've got the technique.

Highlighting The Differences The AI Made

Okay, wow. This is so friggn insightful. Visually scanning down, I can tell a lot of things. First, the horizontal scrolling is out of hand. I'm going to have to split some lines and vertically stack a few things for readability, especially when sharing here online. I can't expect the users to scroll horizontally to see code examples that run long to the right.
Second is that there's the field name in get_suggestion() needs to be inferred from step_id and not hardwired. Okay, done. Ugh, so here is the penalty of the WET approach. Changes like this should be made in every file because there's no inheritance from a superclass. This is where it's painful, with these little tiny global refinements, but luckily, no code has propagated yet.
The changes are so small to have customized it into its first variation, that I can just go ahead and delete the hello_webpage.py variation while I make these global template changes, and then recreate it later using the very easy to spot changes below that still apply.

```
import asyncio
from collections import namedtuple
from datetime import datetime
from urllib.parse import urlparse  # Added for hostname extraction
import webbrowser  # Added for browser pop-up

from fasthtml.common import *
from loguru import logger

"""
Workflow Template

This file demonstrates the basic pattern for Pipulate workflows:
1. Define steps with optional transformations
2. Each step collects or processes data
3. Data flows from one step to the next
4. Steps can be reverted and the workflow can be finalized

To create your own workflow:
1. Copy this file and rename the class, APP_NAME, DISPLAY_NAME, ENDPOINT_MESSAGE
2. Create a training.md file in the training folder (no path needed) and set TRAINING_PROMPT to refer to it
3. Define your own steps
4. Implement custom validation and processing as needed
"""

# Each step represents one cell in our linear workflow.
Step = namedtuple('Step', ['id', 'done', 'show', 'refill', 'transform'], defaults=(None,))
```

```
class HelloFlow:WebFlow:
    APP_NAME = "hello""web"
    DISPLAY_NAME = "Hello World"Webpage"
    ENDPOINT_MESSAGE = "This simple workflow demonstratesopens a basic Hello World example.webpage in your browser. Enter an ID to start or resume your workflow."
    TRAINING_PROMPT = "workflow_template.md""web_flow.md"  # Can refer to markdown in the training folder (no path needed)
    PRESERVE_REFILL = True

    def __init__(self, app, pipulate, pipeline, db, app_name=APP_NAME):
        self.app = app
        self.app_name = app_name
        self.pipulate = pipulate
        self.pipeline = pipeline
        self.db = db
        pip = self.pipulate

        steps = [
            # Define the ordered sequence of workflow steps
            Step(id='step_01', done='name', show='Your Name',done='url', show='Website Address', refill=True),
            Step(id='step_02', done='greeting', show='Hello Message',done='domain', show='Domain', refill=False, transform=lambda name: f"Hello {name}"),url: urlparse(url).hostname or ""),
            Step(id='finalize', done='finalized', show='Finalize', refill=False)
        ]

        self.steps = steps
        self.steps_indices = {step.id: i for i, step in enumerate(steps)}

        # Define messages for each step
        self.step_messages = {
            "new": "Enter an ID to begin.",
            "finalize": {
                "ready": "All steps complete. Ready to finalize workflow.",
                "complete": "Workflow finalized. Use Unfinalize to make changes."
            }
        }

        # For each non-finalize step, an input and completion message is automatically generated
        for step in steps:
            if step.done != 'finalized':
                self.step_messages[step.id] = {
                    "input": f"{pip.fmt(step.id)}: Please enter {step.show}.",
                    "complete": f"{step.show} complete. Continue to next step."
                }
```

```
    # Define routes for all workflow methods.
    routes = [
        # These are the standard routes for all workflows
        (f"/{app_name}", self.landing),
        (f"/{app_name}/init", self.init, ["POST"]),
        (f"/{app_name}/jump_to_step", self.jump_to_step, ["POST"]),
        (f"/{app_name}/revert", self.handle_revert, ["POST"]),
        (f"/{app_name}/finalize", self.finalize, ["GET", "POST"]),
        (f"/{app_name}/unfinalize", self.unfinalize, ["POST"]),

        # Individual step_xx and step_xx_submit routes per step (except finalize)
        (f"/{app_name}/step_01", self.step_01),
        (f"/{app_name}/step_01_submit", self.step_01_submit, ["POST"]),
        (f"/{app_name}/step_02", self.step_02),
        (f"/{app_name}/step_02_submit", self.step_02_submit, ["POST"]),
    ]
    # Register the routes
    for path, handler, *methods in routes:
        method_list = methods[0] if methods else ["GET"]
        self.app.route(path, methods=method_list)(handler)

async def landing(self):
    pip, pipeline, steps, app_name = self.pipulate, self.pipeline, self.steps, self.app_name
    title = f"{self.DISPLAY_NAME or app_name.title()}: {len(steps) - 1} Steps + Finalize"
    pipeline.xtra(app_name=app_name)
    existing_ids = [record.url for record in pipeline()]
    return Container(
        Card(
            H2(title),
            P("Enter or resume a Pipeline ID:"),
            Form(
                pip.wrap_with_inline_button(
                    Input(
                        placeholder="🗝 Old or existing ID here",
                        name="pipeline_id",
                        list="pipeline-ids",
                        type="text",
                        required=True,
                        autofocus=True,
                    ),
                    button_label=f"Start {self.DISPLAY_NAME} 🔑",
                    button_class="secondary"
                ),
                Datalist(*[Option(value=pid) for pid in existing_ids], id="pipeline-ids"),
                hx_post=f"/{app_name}/init",
                hx_target=f"#{app_name}-container"
            )
        ),
```

```
            Div(id=f"{app_name}-container")
        )

    async def init(self, request):
        pip, db, steps, app_name = self.pipulate, self.db, self.steps, self.app_name
        form = await request.form()
        pipeline_id = form.get("pipeline_id", "untitled")
        db["pipeline_id"] = pipeline_id
        state, error = pip.initialize_if_missing(pipeline_id, {"app_name": app_name})
        if error:
            return error

        # After loading the state, check if all steps are complete
        all_steps_complete = True
        for step in steps[:-1]:  # Exclude finalize step
            if step.id not in state or step.done not in state[step.id]:
                all_steps_complete = False
                break

        # Check if workflow is finalized
        is_finalized = "finalize" in state and "finalized" in state["finalize"]

        # Add information about the workflow ID to conversation history
        id_message = f"Workflow ID: {pipeline_id}. You can use this ID to return to this workflow
later."
        await pip.stream(id_message)

        # Add a small delay to ensure messages appear in the correct order
        await asyncio.sleep(0.5)

        # If all steps are complete, show an appropriate message
        if all_steps_complete:
            if is_finalized:
                await pip.stream(f"Workflow is complete and finalized. Use Unfinalize to make
changes.")
            else:
                await pip.stream(f"Workflow is complete but not finalized. Press Finalize to lock your
data.")
        else:
            # If it's a new workflow, add a brief explanation
            if not any(step.id in state for step in self.steps):
                await pip.stream("Please complete each step in sequence. Your progress will be
saved automatically.")

        # Add another delay before loading the first step
        await asyncio.sleep(0.5)

        placeholders = self.generate_step_placeholders(steps, app_name)
```

```
    return Div(*placeholders, id=f"{app_name}-container")

async def step_01(self, request):
    pip, db, steps, app_name = self.pipulate, self.db, self.steps, self.app_name
    step_id = "step_01"
    step_index = self.steps_indices[step_id]
    step = steps[step_index]
    next_step_id = steps[step_index + 1].id if step_index < len(steps) - 1 else None
    pipeline_id = db.get("pipeline_id", "unknown")
    state = pip.read_state(pipeline_id)
    step_data = pip.get_step_data(pipeline_id, step_id, {})
    user_val = step_data.get(step.done, "")

    if step.done == 'finalized':
        finalize_data = pip.get_step_data(pipeline_id, "finalize", {})
        if "finalized" in finalize_data:
            return Card(
                H3("Pipeline Finalized"),
                P("All steps are locked."),
                Form(
                    Button("Unfinalize", type="submit", style=pip.get_style("warning_button")),
                    hx_post=f"/{app_name}/unfinalize",
                    hx_target=f"#{app_name}-container",
                    hx_swap="outerHTML"
                )
            )
        else:
            return Div(
                Card(
                    H3("Finalize Pipeline"),
                    P("You can finalize this pipeline or go back to fix something."),
                    Form(
                        Button("Finalize All Steps", type="submit"),
                        hx_post=f"/{app_name}/finalize",
                        hx_target=f"#{app_name}-container",
                        hx_swap="outerHTML"
                    )
                ),
                id=step_id
            )

    finalize_data = pip.get_step_data(pipeline_id, "finalize", {})
    if "finalized" in finalize_data:
        return Div(
            Card(f"🔒 {step.show}: {user_val}"),
            Div(id=next_step_id, hx_get=f"/{self.app_name}/{next_step_id}", hx_trigger="load")
        )
```

```
        if user_val and state.get("_revert_target") != step_id:
            return Div(
                pip.revert_control(step_id=step_id, app_name=app_name, message=f"{step.show}:
{user_val}", steps=steps),
                Div(id=next_step_id, hx_get=f"/{app_name}/{next_step_id}", hx_trigger="load")
            )
        else:
            display_value = user_val if (step.refill and user_val and self.PRESERVE_REFILL) else
await self.get_suggestion(step_id, state)

            await pip.stream(self.step_messages[step_id]["input"])
            return Div(
                Card(
                    H3(f"{pip.fmt(step.id)}: Enter {step.show}"),
                    Form(
                        pip.wrap_with_inline_button(
                            Input(
                                type="text",
                                name=step.done,
                                value=display_value,
                                placeholder=f"Enter {step.show}",
                                required=True,
                                autofocus=True
                            )
                        ),
                        hx_post=f"/{app_name}/{step.id}_submit",
                        hx_target=f"#{step.id}"
                    )
                ),
                Div(id=next_step_id),
                id=step.id
            )

    async def step_01_submit(self, request):
        pip, db, steps, app_name = self.pipulate, self.db, self.steps, self.app_name
        step_id = "step_01"
        step_index = self.steps_indices[step_id]
        step = steps[step_index]
        pipeline_id = db.get("pipeline_id", "unknown")
        if step.done == 'finalized':
            state = pip.read_state(pipeline_id)
            state[step_id] = {step.done: True}
            pip.write_state(pipeline_id, state)
            message = await pip.get_state_message(pipeline_id, steps, self.step_messages)
            await pip.stream(message)
            placeholders = self.generate_step_placeholders(steps, app_name)
            return Div(*placeholders, id=f"{app_name}-container")
```

```
        form = await request.form()
        user_val = form.get(step.done, "")

        # VALIDATION: Add step-specific validation here
        is_valid = True
        error_msg = ""
        # Example validation: Check if name is not empty
        #Ensure it’s a valid URL-ish string
        if not user_val.strip():#
            is_valid = False#
            error_msg = "Name"Website Address cannot be empty"

        if not is_valid:
            return P(error_msg, style=pip.get_style("error"))

        # PROCESSING: Add step-specific processing here
        processed_val = user_val
        # Example processing: Capitalize name
        #Add https:// if no scheme is provided
        if not processed_val.startswith(('http://', 'https://')):
            processed_val = user_val.capitalize()f"https://{processed_val}"
        # Open the browser immediately
        webbrowser.open(processed_val)

        next_step_id = steps[step_index + 1].id if step_index < len(steps) - 1 else None
        await pip.clear_steps_from(pipeline_id, step_id, steps)

        state = pip.read_state(pipeline_id)
        state[step_id] = {step.done: processed_val}
        if "_revert_target" in state:
            del state["_revert_target"]
        pip.write_state(pipeline_id, state)

        # Send the value confirmation
        await pip.stream(f"{step.show}: {processed_val}")

        # If this is the last regular step (before finalize), add a prompt to finalize
        if next_step_id == "finalize":
            await asyncio.sleep(0.1)  # Small delay for better readability
            await pip.stream("All steps complete! Please press the Finalize button below to save
your data.")

        return Div(
            pip.revert_control(step_id=step_id, app_name=app_name, message=f"{step.show}:
{processed_val}", steps=steps),
            Div(id=next_step_id, hx_get=f"/{app_name}/{next_step_id}", hx_trigger="load")
        )
```

```
    async def step_02(self, request):
        pip, db, steps, app_name = self.pipulate, self.db, self.steps, self.app_name
        step_id = "step_02"
        step_index = self.steps_indices[step_id]
        step = steps[step_index]
        next_step_id = steps[step_index + 1].id if step_index < len(steps) - 1 else None
        pipeline_id = db.get("pipeline_id", "unknown")
        state = pip.read_state(pipeline_id)
        step_data = pip.get_step_data(pipeline_id, step_id, {})
        user_val = step_data.get(step.done, "")

        if step.done == 'finalized':
            finalize_data = pip.get_step_data(pipeline_id, "finalize", {})
            if "finalized" in finalize_data:
                return Card(
                    H3("Pipeline Finalized"),
                    P("All steps are locked."),
                    Form(
                        Button("Unfinalize", type="submit", style=pip.get_style("warning_button")),
                        hx_post=f"/{app_name}/unfinalize",
                        hx_target=f"#{app_name}-container",
                        hx_swap="outerHTML"
                    )
                )
            else:
                return Div(
                    Card(
                        H3("Finalize Pipeline"),
                        P("You can finalize this pipeline or go back to fix something."),
                        Form(
                            Button("Finalize All Steps", type="submit",
style=pip.get_style("primary_button")),
                            hx_post=f"/{app_name}/finalize",
                            hx_target=f"#{app_name}-container",
                            hx_swap="outerHTML"
                        )
                    ),
                    id=step_id
                )

        finalize_data = pip.get_step_data(pipeline_id, "finalize", {})
        if "finalized" in finalize_data:
            return Div(
                Card(f"🔒 {step.show}: {user_val}"),
                Div(id=next_step_id, hx_get=f"/{self.app_name}/{next_step_id}", hx_trigger="load")
            )

        if user_val and state.get("_revert_target") != step_id:
```

```
            return Div(
                pip.revert_control(step_id=step_id, app_name=app_name, message=f"{step.show}:
{user_val}", steps=steps),
                Div(id=next_step_id, hx_get=f"/{app_name}/{next_step_id}", hx_trigger="load")
            )
        else:
            display_value = user_val if (step.refill and user_val and self.PRESERVE_REFILL) else
await self.get_suggestion(step_id, state)

            await pip.stream(self.step_messages[step_id]["input"])
            return Div(
                Card(
                    H3(f"{pip.fmt(step.id)}: Enter {step.show}"),
                    Form(
                        pip.wrap_with_inline_button(
                            Input(
                                type="text",
                                name=step.done,
                                value=display_value,
                                placeholder=f"Enter {step.show}",
                                required=True,
                                autofocus=True
                            )
                        ),
                        hx_post=f"/{app_name}/{step.id}_submit",
                        hx_target=f"#{step.id}"
                    )
                ),
                Div(id=next_step_id),
                id=step.id
            )

    async def step_02_submit(self, request):
        pip, db, steps, app_name = self.pipulate, self.db, self.steps, self.app_name
        step_id = "step_02"
        step_index = self.steps_indices[step_id]
        step = steps[step_index]
        pipeline_id = db.get("pipeline_id", "unknown")
        if step.done == 'finalized':
            state = pip.read_state(pipeline_id)
            state[step_id] = {step.done: True}
            pip.write_state(pipeline_id, state)
            message = await pip.get_state_message(pipeline_id, steps, self.step_messages)
            await pip.stream(message)
            placeholders = self.generate_step_placeholders(steps, app_name)
            return Div(*placeholders, id=f"{app_name}-container")

        form = await request.form()
```

```
        user_val = form.get(step.done, "")

        # VALIDATION: Add step-specific validation here
        is_valid = True
        error_msg = ""
        # Example validation: Check if namedomain is not empty#
        if not user_val.strip():#
            is_valid = False#
            error_msg = "Name"Domain cannot be empty"

        if not is_valid:
            return P(error_msg, style=pip.get_style("error"))

        # PROCESSING: Add step-specific processing here
        processed_val = user_val
        # Example processing: Capitalize name
        # processed_val = user_val.capitalize()

        next_step_id = steps[step_index + 1].id if step_index < len(steps) - 1 else None
        await pip.clear_steps_from(pipeline_id, step_id, steps)

        state = pip.read_state(pipeline_id)
        state[step_id] = {step.done: processed_val}
        if "_revert_target" in state:
            del state["_revert_target"]
        pip.write_state(pipeline_id, state)

        # Send the value confirmation
        await pip.stream(f"{step.show}: {processed_val}")

        # If this is the last regular step (before finalize), add a prompt to finalize
        if next_step_id == "finalize":
            await asyncio.sleep(0.1)  # Small delay for better readability
            await pip.stream("All steps complete! Please press the Finalize button below to save
your data.")

        return Div(
            pip.revert_control(step_id=step_id, app_name=app_name, message=f"{step.show}:
{processed_val}", steps=steps),
            Div(id=next_step_id, hx_get=f"/{app_name}/{next_step_id}", hx_trigger="load")
        )

    # --- Finalization & Unfinalization ---
    async def finalize(self, request):
        pip, db, steps, app_name = self.pipulate, self.db, self.steps, self.app_name
        pipeline_id = db.get("pipeline_id", "unknown")
        finalize_step = steps[-1]
        finalize_data = pip.get_step_data(pipeline_id, finalize_step.id, {})
```

```
logger.debug(f"Pipeline ID: {pipeline_id}")
logger.debug(f"Finalize step: {finalize_step}")
logger.debug(f"Finalize data: {finalize_data}")

if request.method == "GET":
    if finalize_step.done in finalize_data:
        logger.debug("Pipeline is already finalized")
        return Card(
            H3("All Cards Complete"),
            P("Pipeline is finalized. Use Unfinalize to make changes."),
            Form(
                Button("Unfinalize", type="submit", style=pip.get_style("warning_button")),
                hx_post=f"/{app_name}/unfinalize",
                hx_target=f"#{app_name}-container",
                hx_swap="outerHTML"
            ),
            id=finalize_step.id
        )

    # Check if all previous steps are complete
    non_finalize_steps = steps[:-1]
    all_steps_complete = all(
        pip.get_step_data(pipeline_id, step.id, {}).get(step.done)
        for step in non_finalize_steps
    )
    logger.debug(f"All steps complete: {all_steps_complete}")

    if all_steps_complete:
        return Card(
            H3("Ready to finalize?"),
            P("All data is saved. Lock it in?"),
            Form(
                Button("Finalize", type="submit", style=pip.get_style("primary_button")),
                hx_post=f"/{app_name}/finalize",
                hx_target=f"#{app_name}-container",
                hx_swap="outerHTML"
            ),
            id=finalize_step.id
        )
    else:
        return Div(P("Nothing to finalize yet."), id=finalize_step.id)
else:
    # This is the POST request when they press the Finalize button
    state = pip.read_state(pipeline_id)
    state["finalize"] = {"finalized": True}
    state["updated"] = datetime.now().isoformat()
    pip.write_state(pipeline_id, state)
```

```
            # Send a confirmation message
            await pip.stream("Workflow successfully finalized! Your data has been saved and
locked.")

            # Return the updated UI
            return Div(*self.generate_step_placeholders(steps, app_name),
id=f"{app_name}-container")

    async def unfinalize(self, request):
        pip, db, steps, app_name = self.pipulate, self.db, self.steps, self.app_name
        pipeline_id = db.get("pipeline_id", "unknown")
        state = pip.read_state(pipeline_id)
        if "finalize" in state:
            del state["finalize"]
        pip.write_state(pipeline_id, state)

        # Send a message informing them they can revert to any step
        await pip.stream("Workflow unfinalized! You can now revert to any step and make
changes.")

        placeholders = self.generate_step_placeholders(steps, app_name)
        return Div(*placeholders, id=f"{app_name}-container")

    def generate_step_placeholders(self, steps, app_name):
        placeholders = []
        for i, step in enumerate(steps):
            trigger = "load" if i == 0 else f"stepComplete-{steps[i - 1].id} from:{steps[i - 1].id}"
            placeholders.append(Div(id=step.id, hx_get=f"/{app_name}/{step.id}", hx_trigger=trigger,
hx_swap="outerHTML"))
        return placeholders

    async def jump_to_step(self, request):
        pip, db, steps, app_name = self.pipulate, self.db, self.steps, self.app_name
        form = await request.form()
        step_id = form.get("step_id")
        db["step_id"] = step_id
        return pip.rebuild(app_name, steps)

    async def get_suggestion(self, step_id, state):
        pip, db, steps = self.pipulate, self.db, self.steps
        # If a transform function exists, use the previous step's output.
        step = next((s for s in steps if s.id == step_id), None)
        if not step or not step.transform:
            return ""
        prev_index = self.steps_indices[step_id] - 1
        if prev_index < 0:
            return ""
        prev_step_id = steps[prev_index].id
```

```
        prev_data = pip.get_step_data(db["pipeline_id"], prev_step_id, {})
        prev_word = prev_data.get("name",prev_data.get("url", "")  # Changed from "name" to "url"
        return step.transform(prev_word) if prev_word else ""

    async def handle_revert(self, request):
        pip, db, steps, app_name = self.pipulate, self.db, self.steps, self.app_name
        form = await request.form()
        step_id = form.get("step_id")
        pipeline_id = db.get("pipeline_id", "unknown")
        if not step_id:
            return P("Error: No step specified", style=pip.get_style("error"))
        await pip.clear_steps_from(pipeline_id, step_id, steps)
        state = pip.read_state(pipeline_id)
        state["_revert_target"] = step_id
        pip.write_state(pipeline_id, state)
        message = await pip.get_state_message(pipeline_id, steps, self.step_messages)
        await pip.stream(message)
        placeholders = self.generate_step_placeholders(steps, app_name)
        return Div(*placeholders, id=f"{app_name}-container")
```

An AI's Take

This article details my journey of needing to visually compare code changes made by AI, moving beyond the standard diff output to a more intuitive, color-coded, single-file view. I explain how the combination of git diff --color-words and ansi2html achieves this, allowing for easy identification of added (green) and removed (red) lines, which is crucial for analyzing and publishing AI-assisted code modifications, as demonstrated by the subsequent code comparison highlighting the evolution of a workflow template.
Here are some ideas for titles and headlines:
Titles:

- Seeing the AI's Changes Clearly: A Better Way to Diff Code
- Beyond +/-: Visualizing AI Code Modifications with Color
- My Solution for Easily Comparing AI-Generated Code
- From diff to Delight: A Color-Coded Code Comparison Technique
- Making Sense of AI Code: A Visual Diff Workflow

Headlines:

- Tired of Confusing Diffs? See AI Code Changes in Living Color!
- Unlock the Power of Visual Code Comparison for AI-Assisted Development
- Stop Scrolling! Get a Single-File, Color-Coded View of Your AI's Code Edits
- The Ultimate Technique for Understanding Your AI's Code Changes
- Level Up Your Code Review: See Exactly What Your AI Has Modified

My opinion of the article is that it provides a practical and valuable solution for developers who are increasingly using AI for code generation. The author clearly articulates the problem with traditional diff tools in this context and offers a well-explained, step-by-step method to achieve a

more intuitive visual comparison. This technique not only aids in understanding the specific changes made by AI but also simplifies the process of reviewing and publishing these modifications. The inclusion of a real-world example further enhances the article's usefulness and makes the described method immediately applicable.